

# Urgent Partial Order Reduction for Extended Timed Automata

Kim G. Larsen, Marius Mikučionis, Marco Muñoz, and Jiří Srba

Department of Computer Science, Aalborg University, Denmark  
{kgl,marius,muniz,srba}@cs.aau.dk

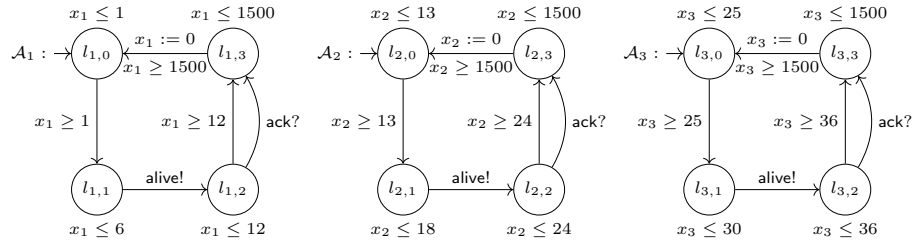
**Abstract.** We propose a partial order reduction method for reachability analysis of networks of timed automata interacting via synchronous channel communication and via shared variables. Our method is based on (classical) symbolic delay transition systems and exploits the urgent behavior of a system, where time does not introduce dependencies among actions. In the presence of urgent behavior in the network, we apply partial order reduction techniques for discrete systems based on stubborn sets. We first describe the framework in the general setting of symbolic delay time transition systems and then instantiate it to the case of timed automata. We implement our approach in the model checker UPPAAL and observe a substantial reduction in the reachable state space for case studies that exhibit frequent urgent behaviour and only a moderate slowdown on models with limited occurrence of urgency.

## 1 Introduction

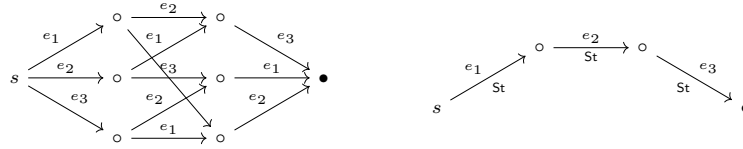
Partial order reduction techniques [4] based on persistent sets [14], ample sets [23] or stubborn sets [17,28] have proved beneficial for the state space exploration of systems that exhibit high degree of concurrency. As many actions in such systems can be (in a syntax-driven manner) considered as independent, these techniques will explore only a subset of the possible interleavings of independent actions while preserving the property of the system we are interested in.

The techniques of partial order reductions for untimed system have only recently been extended to *timed* systems with indication of success. For more than two decades timed systems have resisted several partial order reduction attempts, largely caused by the fact that time introduces additional dependencies between actions that will normally be considered as independent. In [15] the authors show a potential for stubborn reductions for networks of timed automata, however using only approximate abstraction approach. A new idea of exploiting urgency in timed systems in order to facilitate efficient partial order reduction appeared in [8] in the context of timed-arc Petri nets with discrete time.

We take the idea of urgency-based [24] partial order reduction one step further and extend the method towards the case of networks of extended timed automata in the UPPAAL style, including handshake and broadcast communication primitives, communication over shared variables as well as a C-like imperative programming language allowing for complex computation over discrete



(a) Three sensors of the simplified fire alarm system



(b) Fragment of the full and reduced transition systems for the system in Figure 1a

Fig. 1: Simplified Fire Alarm System

structured variables. Our main contribution is a partial order reduction method for urgent behavior based on the classical (zone-based) symbolic semantics for networks of timed automata and its efficient implementation in the industrial strength real-time verification tool UPPAAL. An additional challenge is to develop static analysis for the rich modeling language of UPPAAL and combine it with symbolic model checking techniques in a sound way. On a number of experiments we show the applicability of the proposed method w.r.t. state-space and time reduction.

*Fire Alarm System Example.* To illustrate the effect of our urgency-based partial order reduction technique, we consider a simplified version of an industrial fire alarm system [11]. The system uses a communication protocol based on the Time Division Multiple Access (TDMA) paradigm, and has over 100 sensors each of them assigned a unique time slot for sending and receiving messages. Figure 1a shows a down-scaled and simplified version of the system with three sensors, each modeled as a timed automaton. Each sensor has its own clock  $x_i$ , with the corresponding TDMA slot modeled by guards in ( $x_i \geq 1500$ ) and invariants ( $x_i \leq 1500$ ). At the end of the TDMA cycle i.e. when  $x_i = 1500$  every sensor resets its clock and goes back to its initial location. Figure 1b left shows the fragment of the reachable transition system starting at the configuration  $s = ((l_{1,3}, l_{2,3}, l_{3,3}), x_1 = x_2 = x_3 = 1500)$  where time progress is disabled due to the invariants  $x_i \leq 1500$ . The transitions are induced by the edges  $e_i = \langle l_{i,3}, \tau, x_i \geq 1500, x_i = 0, l_{i,0} \rangle$ . States of the form  $\circ$  denote the so-called zero time states where time cannot progress, whereas the filled state  $\bullet$  denotes a situation where time can delay. Figure 1b right shows the corresponding reduced transition

system that contains only one interleaving sequence that allows us to reach the state where time can delay again.

*Related Work.* The most related work in [8] presents an urgent partial order reduction method for *discrete* time systems based on stubborn set construction [17,28]. The method is instantiated to timed-arc Petri nets and compared to our case, it does not consider discrete data structures nor any communication primitives. In our work we focus on *continuous* time systems modeled as networks of timed automata, requiring us to use symbolic transition system as the underlying semantic model. The idea of applying partial order reduction for independent events that happen at the same time also appeared in [9] however this methods is not as efficient as ours because it is static (precomputed before the state space exploration). In our approach we apply a dynamic reduction that on-the-fly identifies independent actions even in the presence of communication between the components, possibly sharing some resources.

Partial order reduction techniques applied to timed automata [2] include the early works [7,20,10] based on the notion of local and global clocks or the concept of covering as generalized dependencies. However, there is not provided any experimental evaluation of the proposed techniques. There exist also techniques based on event zones [19,22] and on merging zones from different interleaved executions [26]. These are exact techniques comparable to approximate convex-hull abstraction which is by now superseded by the exact LU-abstraction [5]. More recently, over-approximative methods based on abstracted zone graphs were also studied in [15]. The main difference is that our approach is an exact method that is applicable directly to the state-of-the-art techniques implemented in UPPAAL.

Finally, *quasi-equal clocks* [16] are clocks for which in all computations their values are equal or if one clock gets reset then a reset must urgently eventually occur also for the other clocks, assuming that resets occur periodically. Reductions using quasi-equal clocks yield exponential savings and have been used to verify a number of industrial systems. However, this approach is based on syntactic transformations and requires a method for detecting quasi-equal clocks [21]. Our approach fully automatizes reductions based on quasi-equal clocks and further generalizes to scenarios where clock resets have irregular reset periods.

## 2 Partial Order Reduction for Symbolic Delays

We describe the general idea of our partial order reduction technique in terms of symbolic delay transition systems. Intuitively a symbolic delay corresponds to time elapsing in the zone graph for timed automata or flow in the region graph of a hybrid system. Let  $A$  be a set of actions and  $\delta$  a symbolic delay with  $A \cap \{\delta\} = \emptyset$ .

**Definition 1 (Symbolic Delay Transition System).** A symbolic delay transition system is a tuple  $(S, s_0, \rightarrow)$  where  $S$  is a set of states,  $s_0 \in S$  is the initial state, and  $\rightarrow \subseteq S \times (A \cup \{\delta\}) \times S$  is the transition relation.

If  $(s, \alpha, s') \in \rightarrow$  we write  $s \xrightarrow{\alpha} s'$ . In this paper we consider only deterministic systems: a transition system is *deterministic* if  $s \xrightarrow{\alpha} s'$  and  $s \xrightarrow{\alpha} s''$  implies  $s' = s''$ . For the rest of this section, let us assume a fixed symbolic delay transition system  $(S, s_0, \rightarrow)$  and a set of goal states  $G \subseteq S$ .

A state  $s \in S$  is *zero time* if it can not delay, denoted by  $\text{zt}(s)$  and defined by  $\text{zt}(s)$  iff  $\forall s' \in S, \alpha \in A \cup \{\delta\}. s \xrightarrow{\alpha} s' \implies \alpha \in A$ . A *reduction* is a function  $\text{St} : S \rightarrow 2^A$ . A reduced transition relation is a relation  $\xrightarrow{\text{St}} \subseteq \rightarrow$  such that  $s \xrightarrow{\alpha} s'$  iff  $s \xrightarrow{\alpha} s'$  and  $\alpha \in \text{St}(s) \cup \{\delta\}$ . For a given state  $s \in S$  we define  $\overline{\text{St}(s)} \stackrel{\text{def}}{=} A \setminus \text{St}(s)$  to be the set of all actions not in  $\text{St}(s)$ . Given a sequence of labels  $w = \alpha_1 \alpha_2 \alpha_3 \dots \alpha_n \in (A \cup \{\delta\})^*$  we write  $s \xrightarrow{w} s'$  iff  $s \xrightarrow{\alpha_1} \dots \xrightarrow{\alpha_n} s'$ . If a sequence  $w$  of length  $n$  is such that  $s \xrightarrow{w} s'$  we also write  $s \rightarrow^n s'$ . The set of *enabled actions* at state  $s \in S$  is  $\text{En}(s) \stackrel{\text{def}}{=} \{a \in A \mid \exists s' \in S. s \xrightarrow{a} s'\}$ .

The *reachability problem*, given a symbolic delay transition system  $(S, s_0, \rightarrow)$  and a set of goal states  $G$ , is to decide whether there is  $s' \in G$  such that  $s_0 \rightarrow^* s'$ .

**Definition 2 (Reachability Preserving Reduction).** *A reduction  $\text{St}$  is reachability preserving if it satisfies the following conditions:*

- ( $\mathcal{Z}$ )  $\forall s \in S. \neg \text{zt}(s) \implies \text{En}(s) \subseteq \text{St}(s)$
- ( $\mathcal{D}$ )  $\forall s, s' \in S. \forall w \in \overline{\text{St}(s)}^*. \text{zt}(s) \wedge s \xrightarrow{w} s' \implies \text{zt}(s')$
- ( $\mathcal{R}$ )  $\forall s, s' \in S. \forall w \in \overline{\text{St}(s)}^*. \text{zt}(s) \wedge s \xrightarrow{w} s' \wedge s \notin G \implies s' \notin G$
- ( $\mathcal{W}$ )  $\forall s, s' \in S. \forall w \in \overline{\text{St}(s)}^*. \forall a \in \text{St}(s). \text{zt}(s) \wedge s \xrightarrow{wa} s' \implies s \xrightarrow{aw} s'$

If a delay is possible at state  $s$  Condition  $\mathcal{Z}$  will ensure that there is no reduction. Condition  $\mathcal{D}$  ensures that states which can delay are preserved. Condition  $\mathcal{R}$  ensures that goal states are preserved and finally Condition  $\mathcal{W}$  corresponds to the classical stubborn set requirement that stubborn actions can be commuted to the beginning of the execution. The following theorem was proved in [8] for the case of timed transitions systems.

**Theorem 1 (Reachability Preservation).** *Let  $\text{St}$  be a reachability preserving reduction. Let  $s \in S$  and  $s \rightarrow^n s'$  for some  $s' \in G$  then  $s \xrightarrow{\text{St}}^m s''$  for some  $s'' \in G$  where  $m \leq n$ .*

### 3 Extended Timed Automata (XTA)

We apply our method to the theory of timed automata [2]. Our formal model is *extended timed automata* and it is an abstract representation of modeling formalism used in the tool UPPAAL [6].

*Clocks and Discrete Variables.* Let  $X$  be a set of *clocks*. A *clock valuation* is a function  $\mu : X \rightarrow \mathbb{R}_{\geq 0}$ . We use  $\mathcal{V}(X)$  to denote the sets of all valuations for clocks in  $X$ . Let  $V$  be a set of *discrete variables*. The function  $D$  assigns to each variable  $v \in V$  a finite domain  $D(v)$ . A *variable valuation* is a function  $\nu : V \rightarrow \bigcup_{v \in V} D(v)$  that maps variables to values such that  $\nu(v) \in D(v)$ . We use  $\mathcal{V}(V)$  to denote the set of all variable valuations. We let  $\mu_0$  resp.  $\nu_0$  to denote the valuation that maps every clock resp. variable to the value 0.

*Expressions.* We use  $\text{expr}$  to denote an expression over  $V$ . We assume that expressions are well typed and for expression  $\text{expr}$  we use  $D(\text{expr})$  to denote its domain. Given a variable valuation  $\nu$  and an expression  $\text{expr}$ , we use  $\text{expr}^\nu \in D(\text{expr})$  to denote the value of  $\text{expr}$  under  $\nu$ . We use  $V(\text{expr}) \in 2^V$  to denote the set of variables in  $\text{expr}$  such that for all  $v \in V(\text{expr})$  and for all  $\nu, \nu' \in \mathcal{V}(V)$  if  $\nu(v) = \nu'(v)$  then  $\text{expr}^\nu = \text{expr}^{\nu'}$ .

*Constraints.* The set  $B(X)$  is the set of *clock constraints* generated by the grammar  $\phi ::= x \bowtie \text{expr} \mid \phi_1 \wedge \phi_2$ , where  $x \in X$ ,  $D(\text{expr})$  is the domain of all natural numbers  $\mathbb{N}$  and  $\bowtie \in \{<, \leq, \geq, >\}$ . The set  $B(V)$  is a set of *Boolean variable constraints* over  $V$ . The set  $B(X, V)$  of constraints comprises  $B(X)$ ,  $B(V)$ , and conjunctions over clock and variable constraints. Given a constraint  $\phi \in B(X, V)$ , we use  $X(\phi)$  to denote the set of clocks in  $\phi$ , and  $V(\phi)$  to denote the set of variables in  $\phi$ . We define the evaluation of a constraint  $\phi \in B(X, V)$  as  $\phi^\nu$  where expressions in  $\phi$  are evaluated under  $\nu$ .

*Updates.* A *clock update* is of the form  $x := \text{expr}$  where  $x \in X$ , and  $D(\text{expr}) = \mathbb{N}$ . A *variable update* is of the form  $v := \text{expr}$  where  $v \in V$  and  $D(v) = D(\text{expr})$ . The set  $U(X, V)$  of *updates* contains all finite, possibly empty sequences of clock and variable updates. Given clock valuation  $\mu \in \mathcal{V}(X)$ , variable valuation  $\nu \in \mathcal{V}(V)$ , and update  $r \in U(X, V)$ , we use  $r^\nu$  to denote the update resulting after evaluating all expressions in  $r$  under  $\nu$ , we use  $X(r)$  to denote the set of clocks in  $r$ , and  $V(r)$  to denote the set of variables in  $r$ . We let  $\llbracket r^\nu \rrbracket : \mathcal{V}(X) \cup \mathcal{V}(V) \rightarrow \mathcal{V}(X) \cup \mathcal{V}(V)$  be a map from valuations to valuations. We use  $\mu[r^\nu]$  to denote the updated clock valuation  $\llbracket r^\nu \rrbracket(\mu)$ . Analogously, for variable valuation  $\nu'$ , we use  $\nu'[r^\nu]$  to denote the updated variable valuation  $\llbracket r^\nu \rrbracket(\nu')$ .

*Channels.* Given a set  $C$  of *channels*, the set  $H(C)$  of synchronizations over channels is generated by the grammar  $h ::= c[\text{expr}]! \mid c[\text{expr}]? \mid \tau$ , where  $c \in C$ ,  $D(\text{expr}) = \mathbb{N}$ , and  $\tau$  represents an internal action. Given a variable valuation  $\nu$ , for synchronization  $h$  of the form  $c[\text{expr}]!$  we use  $h^\nu$  to denote  $c[\text{expr}^\nu]!$ , and similar for synchronizations of the form  $c[\text{expr}]?$ .

**Definition 3 (Extended Timed Automata XTA).** A extended timed automaton  $\mathcal{A}$  is a tuple  $(L, L^u, L^c, l_0, X, V, H(C), E, I)$  where:  $L$  is a set of locations,  $L^u \subseteq L$  denotes the set of urgent locations in  $L$ ,  $L^c \subseteq L$  denotes the set of committed locations in  $L$  and  $L^u \cap L^c = \emptyset$ ,  $l_0 \in L$  is the initial location,  $X$  is a nonempty the set of clocks,  $V$  is the set of variables,  $H(C)$  is a set of channels expressions for set of channels  $C$ ,  $E \subseteq L \times H(C) \times B(X) \times B(V) \times U(X, V) \times L$  is a set of edges between locations with a channel expressions, a clock guard, a variable guard, an update set, and  $I : L \rightarrow B(X)$  assigns clock invariants to locations.

**Definition 4 (Network of XTA).** A network  $\mathcal{N}$  of XTA consists of a finite sequence  $\mathcal{A}_1, \dots, \mathcal{A}_n$  of XTA, where  $\mathcal{A}_i = (L_i, L_i^u, L_i^c, l_i^0, X_i, V_i, H(C)_i, E_i, I_i)$  for  $1 \leq i \leq n$ . Locations are pairwise disjoint i.e.  $L_i \cap L_j = \emptyset$  for  $1 \leq i, j \leq n$

and  $i \neq j$ . The set of locations is  $L = \cup_{i=1}^n L_i$ , analogously for urgent  $L^u$  and committed  $L^c$  locations. The set of clocks is  $X = \cup_{i=1}^n X_i$  and the set of variables is  $V = \cup_{i=1}^n V_i$ . The set of channel expressions is  $H(C) = \cup_{i=1}^n H(C)_i$ . The set of edges is  $E = \cup_{i=1}^n E_i$ . A location vector is a vector  $\vec{l} = (l_1, \dots, l_n)$ , and  $\vec{l}_0 = (l_1^0, \dots, l_n^0)$  is the initial location vector. The invariant function over location vectors is  $I(\vec{l}) = \bigwedge_i I_i(l_i)$ .

We write  $\vec{l}[l'_i/l_i]$  to denote the vector where the  $i$ -th element  $l_i$  of  $\vec{l}$  is replaced by  $l'_i$ . We write  $\vec{l}^i$  to denote the  $i$ -th element of  $\vec{l}$ .

*Zones.* We assume the canonical satisfaction relation “ $\models$ ” between valuations and constraints in  $B(X)$  and  $B(V)$ . The set  $B^+(X)$  of *extended clock constraints* is generated by the grammar  $\phi ::= x \bowtie c \mid \phi_1 \wedge \phi_2 \mid x - y \bowtie c$ , where  $x, y \in X$ ,  $c \in \mathbb{N}$  and  $\bowtie \in \{<, \leq, \geq, >\}$ . A *zone*  $\llbracket Z \rrbracket$  is a set of clock valuations described by an extended clock constraint  $Z \in B^+(X)$  where  $\llbracket Z \rrbracket \stackrel{\text{def}}{=} \{\mu \in \mathcal{V}(X) \mid \mu \models Z\}$ . When it is clear from the context, we use  $Z$  and  $\llbracket Z \rrbracket$  interchangeably. We define  $Z^\uparrow \stackrel{\text{def}}{=} \{\mu + d \mid \mu \in Z, d \in \mathbb{R}_{\geq 0}\}$ , where for  $d \in \mathbb{R}_{\geq 0}$ ,  $\mu + d$  maps each clock  $x \in X$  to the value  $\mu(x) + d$ . For zone  $Z$  and update  $r$  we define  $Z[r] \stackrel{\text{def}}{=} \{\mu[r] \mid \mu \in Z\}$ .

For timed automata we consider the set of actions  $A = 2^E$  that corresponds to the discrete transitions induced by the edges  $E$ , and  $\delta$  is the delay action induced by non-zero delay transitions. We can now define the symbolic semantics of networks of timed automata in terms of a zone graph (see e.g. [1]).

**Definition 5 (Semantics of a Network of XTA).** Let  $\mathcal{N} = \mathcal{A}_1, \dots, \mathcal{A}_n$  be a network of TA. Its semantics is defined as a symbolic delay transition system (zone graph)  $(S, s_0, \rightarrow)$ , where  $S \subseteq (L_i \times \dots \times L_n) \times B^+(X) \times \mathcal{V}(V)$  is the set of states comprising a location vector, a zone, and a variable valuation,  $s_0 = (\vec{l}_0, \{\mu_0\}, \nu_0)$  is the initial state, and  $\rightarrow \subseteq S \times (A \cup \{\delta\}) \times S$  is the transition relation defined by:

- *delay transition*,  $(\vec{l}, Z, \nu) \xrightarrow{\delta} (\vec{l}, Z^\uparrow \wedge I(\vec{l})^\nu, \nu)$  if  $\vec{l}^i \notin L_i^u \cup L_i^c$  for  $1 \leq i \leq n$ , and  $\exists \mu \in Z, d \in \mathbb{R}_{\geq 0}. d > 0 \wedge \mu + d \models I(\vec{l})^\nu$ ,
- *internal transition*,  $(\vec{l}, Z, \nu) \xrightarrow{\{e_i\}} (\vec{l}[l'_i/l_i], Z', \nu')$  if  $e_i = (l_i, \tau, \phi, \psi, r, l'_i) \in E_i$  s.t.  $Z' = (Z \wedge I(\vec{l})^\nu \wedge \phi^\nu)[r^\nu] \wedge I(\vec{l}[l'_i/l_i])^{\nu'}$ , where  $Z' \neq \emptyset$ ,  $\nu' = \nu[r^\nu]$ ,  $\nu \models \psi^\nu$ , and if  $\vec{l}^k \in L_k^c$  for some  $1 \leq k \leq n$  then  $l_i \in L_i^c$ ,
- *handshake transition*,  $(\vec{l}, Z, \nu) \xrightarrow{\{e_i, e_j\}} (\vec{l}[l'_j/l_j, l'_i/l_i], Z', \nu')$  if there exists  $e_i = (l_i, h_i!, \phi_i, \psi_i, r_i, l'_i) \in E_i$  and  $e_j = (l_j, h_j?, \phi_j, \psi_j, r_j, l'_j) \in E_j$  s.t.  $h_i^\nu = h_j^\nu$ , and  $Z' = (Z \wedge I(\vec{l})^\nu \wedge \phi_i^\nu \wedge \phi_j^\nu)[r_i^\nu][r_j^\nu] \wedge I(\vec{l}[l'_j/l_j, l'_i/l_i])^{\nu'}$ , where  $Z' \neq \emptyset$ ,  $\nu \models (\psi_i^\nu \wedge \psi_j^\nu)$ ,  $\nu' = \nu[r_i^\nu][r_j^\nu]$ , and if  $\vec{l}^k \in L_k^c$  for some  $1 \leq k \leq n$  then  $l_i \in L_i^c$  or  $l_j \in L_j^c$ .

In the following, we are given a network of TA  $\mathcal{N} = \mathcal{A}_1, \dots, \mathcal{A}_n$  with locations  $L$ , clocks  $X$ , variables  $V$ , and induced symbolic transition system  $(S, s_0, \rightarrow)$ .

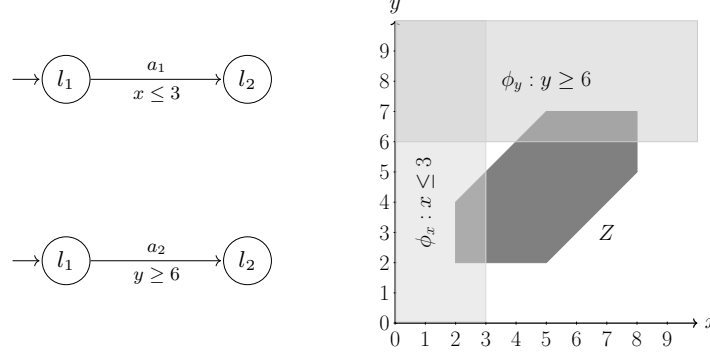


Fig. 2: Two components with actions  $a_1$  and  $a_2$ . Actions are enabled at zone  $Z$ . Note that executing  $a_1$  will disable  $a_2$  and vice versa.

**Definition 6 (Properties).** A formula is given by the grammar  $\phi ::= \text{deadlock} \mid l \mid x \bowtie c \mid \psi_v \mid \phi_1 \wedge \phi_2$ , where  $l \in L$ ,  $x \in X$ ,  $\bowtie \in \{<, \leq, \geq, >\}$ ,  $c \in \mathbb{N}$ , and  $\psi_v$  is a Boolean constraint for  $v \in V$ . Let  $(\vec{l}, Z, \nu) \in S$  be a state. The satisfaction of a formula is inductively defined as follows:

$$\begin{aligned}
 (\vec{l}, Z, \nu) \models \text{deadlock} & \text{ iff } \exists \mu \in Z, \forall d \in \mathbb{R}_{\geq 0}. \text{En}((\vec{l}, \{\mu + d\}, \nu)) = \emptyset \\
 (\vec{l}, Z, \nu) \models l & \text{ iff } \vec{l}^i = l \text{ for some } i \text{ with } 1 \leq i \leq n \\
 (\vec{l}, Z, \nu) \models x \bowtie c & \text{ iff } \exists \mu \in Z. \mu \models x \bowtie c \\
 (\vec{l}, Z, \nu) \models \psi_v & \text{ iff } \nu \models \psi_v \\
 (\vec{l}, Z, \nu) \models \phi \wedge \psi & \text{ iff } (\vec{l}, Z, \nu) \models \phi \text{ and } (\vec{l}, Z, \nu) \models \psi
 \end{aligned}$$

A network satisfies  $\phi$  iff its initial state can reach a state that satisfies  $\phi$ .

## 4 Reachability Preserving Reduction for XTA

In this section we provide syntactic based sound approximations for all the elements required by our technique. In Subsection 4.1 we give a semantic definition for independence of actions, then we describe a syntactic independence relation. In Subsection 4.2 we identify the relevant actions which need to be included in the stubborn set to preserve states which can delay. Finally, in Subsection 4.3 we describe the stubborn sets for preserving goal states. For the rest of this section we are given a network  $\mathcal{N} = \mathcal{A}_1, \dots, \mathcal{A}_n$ , with edges  $E$  and components  $\mathcal{A}_i = (L_i, L_i^u, L_i^c, l_{i_0}, X_i, V_i, H(C)_i, E_i, I_i)$ , the corresponding transition system  $(S, s_0, \rightarrow)$  with actions  $A = 2^E$ , and state  $s = (\vec{l}, Z, \nu)$ .

### 4.1 Independence for Actions

The notion of *independence of actions* plays a key role in partial order reduction. Intuitively two actions are independent if they can not disable each other and they commute.

**Definition 7 (Independence of Actions).** *An independence relation for state  $s \in S$  is a symmetric, anti-reflexive relation  $\rightsquigarrow_s \subseteq A \times A$  satisfying the following conditions for each  $(a_1, a_2) \in \rightsquigarrow_s$ :*

1.  $\forall s' \in S. s \xrightarrow{a_1} s' \wedge a_2 \in \text{En}(s) \implies a_2 \in \text{En}(s')$
2.  $a_1 \in \text{En}(s) \wedge a_2 \in \text{En}(s) \implies \exists s' \in S. s \xrightarrow{a_1 a_2} s' \wedge s \xrightarrow{a_2 a_1} s'$

*If  $(a_1, a_2) \in \rightsquigarrow_s$  they are independent at  $s$  denoted by  $a_1 \rightsquigarrow_s a_2$ . Otherwise they are dependent at  $s$  denoted by  $a_1 \rightsquigarrow_s a_2$ .*

In what follows we will provide a syntactic independence relation on actions. Toward this goal, first we define operations on actions and we define a syntactic independence relation on operations.

*Additional Notation.* For a given edge  $e = (l, h, \phi, \psi, r, l') \in E$  we use  $\text{src}(e)$ ,  $\text{dst}(e)$  to denote the source location  $l$  and the destination location  $l'$  of edge  $e$ . Given actions  $a, a' \in A$ , for action  $a$  we define its *preset* as  $\text{Pre}(a) \stackrel{\text{def}}{=} \{ \text{src}(e) \in L \mid e \in a \}$ , and its *postset* as  $\text{Post}(a) \stackrel{\text{def}}{=} \{ \text{dst}(e) \in L \mid e \in a \}$ . We use  $\text{Active}(a) \stackrel{\text{def}}{=} \{ \mathcal{A}_i \mid \mathcal{A}_i \text{ is in } \mathcal{N} \text{ and } \exists l \in \text{Pre}(a). l \in L_i \}$  to denote the active components for  $a$ . We use  $\text{Parallel}(a, a') \stackrel{\text{def}}{=} \text{Active}(a) \cap \text{Active}(a') = \emptyset$  to denote that actions  $a$  and  $a'$  correspond to different components. For convenience we define *Operations for Actions in TA*. The set of all operations is the set containing all constraints and resets i.e.  $\text{Op}$  is the power set of  $B(X, V) \cup U(X, V)$ . The set of *operations* for action  $a \in A$  is given by,  $\text{Op}(a) \stackrel{\text{def}}{=} \text{Guard}(a) \cup \text{Update}(a)$ . Where the set of guards is  $\text{Guard}(a) \stackrel{\text{def}}{=} \bigcup \{ \phi \wedge \psi \wedge I(l) \wedge I(l') \mid (l, h, \phi, \psi, r, l') \in a \}$ , and the set of updates is  $\text{Update}(a) \stackrel{\text{def}}{=} \bigcup \{ r \mid (l, h, \phi, \psi, r, l') \in a \}$ . Given an operation  $\text{op} \in \text{Op}$ , the set of variables which  $\text{op}$  increments is given by  $\text{Inc}(\text{op}) = \{ v \in V(\text{op}) \mid \exists r \in \text{op} \text{ and } r \text{ includes } v := v + 1 \text{ with } D(v) = \mathbb{N} \}$ . Analogously the set  $\text{Dec}(\text{op})$  contains the variables which  $\text{op}$  decrements  $\text{Dec}(\text{op}) = \{ v \in V(\text{op}) \mid \exists r \in \text{op} \text{ and } r \text{ includes } v := v - 1 \text{ with } D(v) = \mathbb{N} \}$ . The clocks and variables the operation writes is given by  $\text{Write}(\text{op}) \stackrel{\text{def}}{=} \bigcup_{r \in \text{op}} \{ xv \mid r \in U(X, V) \text{ and } xv := \text{expr} \text{ is in } r \}$ , where  $\text{op}'$  is obtained from  $\text{op}$  by removing increment and decrement updates, formally  $\text{op}' = \text{op} \setminus \{ xv := \text{expr} \in \text{op} \mid \text{expr} \text{ is of the form } xv + 1 \text{ or } xv - 1 \}$ . The set  $\text{Readleq}(\text{op}) = \{ xv \in X \cup V \mid xv \leq \text{expr} \in \text{op} \text{ or } xv < \text{expr} \in \text{op} \}$  is the set containing clock and variables which appear in less and equal comparisons. Analogously the set  $\text{Readgeq}(\text{op})$  contains clock and variables which appear in greater and equal comparisons in  $\text{op}$ . The clocks and variables the operation reads is given by  $\text{Read}(\text{op}) \stackrel{\text{def}}{=} X(\text{op}') \cup V(\text{op}')$  where  $\text{op}'$  is obtained from  $\text{op}$  by removing less (greater) and equal comparisons, formally  $\text{op}' = \text{op} \setminus \{ xv \bowtie \text{expr} \in \text{op} \mid \bowtie \in \{ \leq, <, >, \geq \} \}$ . Note that given a zone, a clock constraint can modify (write to) other clocks. Finally  $\Gamma_x(Z) \stackrel{\text{def}}{=} \{ \mu(x) \mid \mu \in Z \}$  is the set of real values for clock  $x$  in zone  $Z$ .

**Definition 8 (Independence of Operations).** *Given operations  $\text{op}_1, \text{op}_2 \in \text{Op}$  and state  $s$ , operation  $\text{op}_1$  is independent of operation  $\text{op}_2$  at  $s$  denoted by*



$\text{op}_1 \not\sim_s^\# \text{op}_2$  iff the following hold:

- (1)  $\text{Read}(\text{op}_1) \cap (\text{Write}(\text{op}_2) \cup \text{Inc}(\text{op}_2) \cup \text{Dec}(\text{op}_2)) = \emptyset$
- (2)  $\text{Readleq}(\text{op}_1) \cap (\text{Write}(\text{op}_2) \cup \text{Inc}(\text{op}_2)) = \emptyset$
- (3)  $\text{Readgeq}(\text{op}_1) \cap (\text{Write}(\text{op}_2) \cup \text{Dec}(\text{op}_2)) = \emptyset$
- (4)  $\text{Write}(\text{op}_1) \cap (\text{Write}(\text{op}_2) \cup \text{Inc}(\text{op}_2) \cup \text{Dec}(\text{op}_2)) = \emptyset$
- (5)  $\text{Inc}(\text{op}_1) \cap (\text{Write}(\text{op}_2) \cup \text{Dec}(\text{op}_2)) = \emptyset$
- (6)  $\text{Dec}(\text{op}_1) \cap (\text{Write}(\text{op}_2) \cup \text{Inc}(\text{op}_2)) = \emptyset$
- (7)  $\{x \mid x \in X(\text{op}_1) \cup X(\text{op}_2) \text{ and } |I_x(Z)| \neq 1 \text{ and } \text{op}_1, \text{op}_2 \in B(X, V)\} = \emptyset$

If  $\text{op}_1 \not\sim_s^\# \text{op}_2$  and  $\text{op}_2 \not\sim_s^\# \text{op}_1$  then we write  $\text{op}_1 \not\rightsquigarrow_s^\# \text{op}_2$  and say that  $\text{op}_1$  and  $\text{op}_2$  are independent at  $s$ . We write  $\text{op}_1 \rightsquigarrow_s^\# \text{op}_2$  iff  $\text{op}_1$  and  $\text{op}_2$  are dependent.

Intuitively two operations are independent if they read and write in different variables, note that increments and decrements are treated specially. Additionally for timed automata we need to consider that applying a guard affects a number of clocks. As an example consider Figure 2, we have that  $Z \cap \phi_x \neq \emptyset$  and  $Z \cap \phi_y \neq \emptyset$ . However, if we apply  $\phi_x$  we have that  $(Z \cap \phi_x) \cap \phi_y = \emptyset$  this will cause the corresponding actions to disable each other. Condition (7) is not satisfied for clocks  $x$  or  $y$  in zone  $Z$ . Therefore we have  $\phi_x \rightsquigarrow_s^\# \phi_y$ . Note that Condition (7) is rather strong, since only zones which are lines or points will satisfy it (which is often the case in urgent states), relaxing this condition is subject of future work. Given two independent operations, we can conclude with a number of rules which are useful for showing that two actions do not disable each other and commute in extended timed automata.

**Lemma 1.** *Given state  $s = (\vec{l}, Z, \nu)$ , constraints  $\phi, \phi' \in B(X)$ , update  $r \in U(X)$ , and variable valuations  $\nu, \nu_1 \in \mathcal{V}(V)$ . The following hold:*

- (1) if  $\phi \not\rightsquigarrow_s^\# r$  then  $\llbracket (Z \wedge \phi^\nu)[r^\nu] \rrbracket = \llbracket Z[r^\nu] \wedge \phi^\nu \rrbracket$
- (2) if  $\llbracket (Z \wedge \phi^\nu)[r^\nu] \wedge \phi^\nu \rrbracket \neq \emptyset$  then  $\llbracket (Z \wedge \phi^\nu)[r^\nu] \wedge \phi^\nu \rrbracket = \llbracket (Z \wedge \phi^\nu)[r^\nu] \rrbracket$
- (3) if  $\phi \not\rightsquigarrow_s^\# \phi'$  and  $\mu \in \llbracket Z \wedge (\phi')^\nu \rrbracket$  and  $\llbracket Z \wedge \phi^{\nu_1} \rrbracket \neq \emptyset$  then  $\mu \in \llbracket (Z \wedge \phi^{\nu_1}) \rrbracket$
- (4) if  $\forall x \in X(\phi). |I_x(Z)| = 1$  and  $\llbracket Z \wedge \phi^\nu \rrbracket \neq \emptyset$  and  $\llbracket Z \wedge \phi^{\nu_1} \rrbracket \neq \emptyset$  then  $\llbracket Z \wedge \phi^\nu \rrbracket = \llbracket Z \wedge \phi^{\nu_1} \rrbracket$

Lemma 1 (1) states that if a reset is independent of a constraint then the reset does not affect the constraint. Lemma 1 (2) does not require independent operations, in our proofs it is used to remove redundant application of invariants from components which are not involved in transitions. Lemma 1 (3) implicitly uses Condition (7) from Definition 8 to show that a valuation satisfying a guard is preserved after applying another guard. Lemma 1 (4) states that if a guard  $\phi$  has been updated (via increment or decrement which in our case always produce a “bigger” constraint), then because of the shape of the zone the intersections will produce the same set.

**Definition 9 (Syntactic Independence of Actions).** *Given a state  $s = (\vec{l}, Z, \nu)$  with  $\vec{l} = (l_1, \dots, l_n)$  and two actions  $a_1, a_2 \in A_s^\#$ . Actions  $a_1$  and  $a_2$  are syntactically independent at state  $s$  denoted by  $a_1 \not\rightsquigarrow_s^\# a_2$  if and only if the*

following conditions hold:

$$\text{(Ind1)} \quad \text{Pre}(a_1) \cap \text{Pre}(a_2) = \emptyset$$

$$\text{(Ind2)} \quad \exists l \in F(a_1). l \in L^c \iff \exists l \in F(a_2). l \in L^c \text{ for } F \in \{\text{Pre}, \text{Post}\}.$$

$$\text{(Ind3)} \quad \forall op_1 \in \text{Op}(a_1), op_2 \in \text{Op}(a_2). op_1 \not\rightsquigarrow_s^\# op_2$$

$$\text{(Ind4)} \quad \forall i \in \{1, 2\}, op \in \text{Op}(a_i), j \in \{1, \dots, n\}. \vec{l}^j \notin \text{Pre}(a_i) \implies op \not\rightsquigarrow_s^\# I(\vec{l}^j)$$

Condition (Ind1) ensures that the source locations for the actions are disjoint. Condition (Ind2) takes into account the semantics of committed locations and prevents actions from disabling each other. Condition (Ind3) ensures that all the operations on the actions are independent. Finally Condition (Ind4) ensures that the operations in actions  $a_i$  for  $i \in \{1, 2\}$  do not modify the invariant of other components which could disable action  $a_{3-i}$ . When these syntactic conditions are satisfied we have the following theorem.

**Theorem 2.** *Given a zero time state  $s \in S$  and two actions  $a_1, a_2 \in A_s^\#$ . If  $a_1 \not\rightsquigarrow_s^\# a_2$  then  $a_1 \not\rightsquigarrow_s a_2$ .*

Our analysis uses the current state  $s = (\vec{l}, Z, \nu)$  to conclude if two actions are independent at  $s$ . In particular we use the zone  $Z$  in Definition 8 Condition (7) to detect clock constraint dependencies. Due to this condition we can make assumptions about the shape of the zone  $Z$  which allow us to conclude that if the actions were syntactically independent at  $s$  then so they are in states reachable via independent actions.

**Corollary 1.** *Given state  $s$ , action  $a \in A_s^\#$ , and  $A' = \{a' \in A_s^\# \mid a \rightsquigarrow_s^\# a'\}$ . Then  $\forall s' \in S. a' \in (A_s^\# \setminus A'), w \in (A_s^\# \setminus A')^*. \text{zt}(s) \wedge s \xrightarrow{w} s' \xrightarrow{a'} s'' \implies a \not\rightsquigarrow_s^\# s', a'$ .*

## 4.2 Preserving Non-Zero Time States

In order to satisfy Condition  $\mathcal{D}$  from Definition 2, which ensures that the reduction preserves states that can delay, we need to include particular actions to the stubborn set. In XTA time can not elapse at an urgent (committed) location or if invariant is stopping time.

**Definition 10 (Time Enabling Action).** *An action  $a \in A$  is a time enabling action at zero time state  $s = (\vec{l}, Z, \nu)$  if executing  $a$  may cause time to elapse. Formally  $\text{tea}^\#(a, s)$  iff  $(\exists l \in \text{Pre}(a). l \in L^u \cup L^c) \vee (\forall \mu \in Z, d \in \mathbb{R}_{\geq 0}. \mu + d \models I(l) \implies d = 0)$ .*

Consider again Figure 1a and the zero time state  $s = ((l_{1,3}, l_{2,3}, l_{3,3}), x_1 = x_2 = x_3 = 1500)$  and actions  $a_i = \{(l_{i,3}, \tau, x_i \geq 1500, x_i = 0, l_{i,0})\}$ . The actions are time enabling actions i.e.  $\text{tea}^\#(a_i, s)$  for  $i \in \{1, 2, 3\}$ . Note that as long as a time enabling action is enabled, time can not elapse. Thus executing independent actions can not cause time to progress.

**Lemma 2.** *Let  $s \in S$ ,  $a \in \text{En}(s)$  with  $\text{tea}^\#(a, s)$  and  $\text{Delay}_s^\# \stackrel{\text{def}}{=} \{a\} \cup \{a' \in A_s^\# \mid a \rightsquigarrow_s^\# a'\}$ . Then  $\forall s' \in S, w \in (A_s^\# \setminus \text{Delay}_s^\#)^*. s \xrightarrow{w} s' \wedge \text{zt}(s) \implies \text{zt}(s')$ .*

### 4.3 Preserving Goal States

In order to satisfy Condition  $\mathcal{R}$  from Definition 2, which ensures that the reduction preserves goal states, we need to include actions whose execution is necessary to reach a goal state.

**Definition 11 (Interesting Actions for Properties).** For formula  $\varphi$  and state  $s$  such that  $s \not\models \varphi$ . The set  $\varphi_s^\sharp \subseteq A_s^\sharp$  is defined recursively based on the structure of  $\varphi$  as given by the following table:

Formula $\varphi$	$\varphi_s^\sharp$
$l$	$\{a \in A_s^\sharp \mid l \in \text{Post}(a)\}$
$\text{deadlock}$	$\text{pick } a \in \text{En}(s) \text{ then } \{a\} \cup \{a' \in A_s^\sharp \mid (\text{Pre}(a) \cap \text{Pre}(a') \neq \emptyset) \vee (\text{Parallel}(a, a') \wedge a \rightsquigarrow_s^\sharp a')\}$
$x \bowtie c$	$\{a \in A_s^\sharp \mid \exists \text{op} \in \text{Update}(a). x \bowtie c \rightsquigarrow_s^\sharp \text{op}\}$
$\varphi_v$ for $v \in V$	$\{a \in A_s^\sharp \mid \exists \text{op} \in \text{Op}(a). \varphi_v \rightsquigarrow_s^\sharp \text{op}\}$
$\varphi_1 \wedge \varphi_2$	$(\varphi_i)_s^\sharp$ for some $i \in \{1, 2\}$ where $s \not\models \varphi_i$

**Lemma 3.** Given a state  $s$ , a formula  $\varphi$ , and the set  $\varphi_s^\sharp$ . Then  $\forall s' \in S, w \in (A_s^\sharp \setminus \varphi_s^\sharp)^*. s \xrightarrow{w} s' \wedge \text{zt}(s) \wedge s \not\models \varphi \implies s' \not\models \varphi$ .

## 5 Computing Stubborn Sets in UPPAAL

We shall first provide a high level algorithm to compute a reachability preserving reduction for networks of timed automata and then discuss details related to the implementation of our technique in the model checker UPPAAL.

### 5.1 Algorithm

Assume a given network of XTA and reachability formula  $\varphi$ . During the reachability analysis, we repeatedly use Algorithm 1 at every generated state  $s$  to compute a reduction  $\text{St}^\sharp$  that satisfies the conditions from Definition 2. At Line 1, we output  $\text{En}(s)$  should the state  $s$  be non-zero time state, thus satisfying Condition  $\mathcal{Z}$ . Line 3 includes all actions that are relevant for the preservation of the reachability of states that can delay or belong to the goal states. Together with Lemma 2 and Lemma 3 this ensures that Condition  $\mathcal{D}$  and Condition  $\mathcal{R}$  are satisfied. Finally, the while loop starting at Line 5 ensures Condition  $\mathcal{W}$ . The while loop considers an action  $a \in \text{St}_s^\sharp$ , if this action is not enabled then it will include all necessary actions which can enable it. This is done by adding actions which modify the location vector at Line 11, or by adding actions which modify the guards in  $a$  at Line 14. In the case where action  $a$  is enabled then the for loop at Line 16 includes all actions that are not independent with  $a$ .

Additionally, note that the set  $A_s^\sharp$  is finite and in each iteration the size of  $\text{St}_s^\sharp$  can only increase because the only operation applied to  $\text{St}_s^\sharp$  is union. In the worst case we have  $\text{St}_s^\sharp = A_s^\sharp$  and hence the algorithm terminates.

**Theorem 3 (Total Correctness).** Let  $\mathcal{N}$  be a network of XTA and  $\varphi$  a formula. Algorithm 1 terminates and  $\text{St}^\sharp$  is a reachability preserving reduction where  $\text{St}^\sharp(s)$  is the output of Algorithm 1 for every state  $s \in S$ .

**Algorithm 1** Computing conditional stubborn sets**Input** Network  $\mathcal{A}_1, \dots, \mathcal{A}_n$ , state  $s = (\vec{l}, Z, \nu)$ , and formula  $\varphi$ .**Output** Conditional stubborn set  $\text{St}_s^\sharp$ 


---

```

1: if  $\neg \text{zt}(s)$  then return  $\text{En}(s)$ ;
2: compute  $A_s^\sharp$  and  $\varphi_s^\sharp$ ;
3: if  $\forall a \in \varphi_s^\sharp. \neg \text{tea}^\sharp(a, s)$  then pick  $a \in \text{En}(s)$  with  $\text{tea}^\sharp(a, s)$ ;  $\varphi^\sharp := \varphi^\sharp \cup \{a\}$ ;
4:  $W := \varphi_s^\sharp$ ;  $R := A_s^\sharp$ ;  $\text{St}_s^\sharp := W$ 
5: while  $W \neq \emptyset$  and  $\text{En}(s) \cap \text{St}_s^\sharp \neq \text{En}(s)$  do
6:   Pick  $a \in W$ ;  $W := W \setminus \{a\}$ ;  $\text{St}_s^\sharp := \text{St}_s^\sharp \cup \{a\}$ ;  $R := R \setminus \{a\}$ ;
7:   if  $a \notin \text{En}(s)$  then
8:     for all  $e \in a$  do
9:       if  $\text{src}(e)$  is not in  $\vec{l}$  then
10:        for all  $a' \in R$  do
11:          if  $\text{src}(e) \in \text{Post}(a')$  then  $W := W \cup \{a'\}$ ;
12:        if exists  $g \in \text{Guard}(\{e\})$  such that  $s \not\models g$  then
13:          for all  $a' \in R$  do
14:            if  $\exists r \in \text{Update}(a'). g \rightsquigarrow_s^\sharp r$  then  $W := W \cup \{a'\}$ ;
15:        if  $a \in \text{En}(s)$  then
16:          for all  $a' \in R$  do
17:            if  $(\text{Pre}(a) \cap \text{Pre}(a') \neq \emptyset) \vee (\text{Parallel}(a, a') \wedge a \rightsquigarrow_s^\sharp a')$  then
18:               $W := W \cup \{a'\}$ ;
19: return  $\text{St}_s^\sharp$ ;

```

---

## 5.2 Implementation Details

Algorithm 1 is inserted as a state successor filter after the state successors are computed. This filter passes through only the states that are the result of stubborn actions. To improve the efficiency, the stubborn set is computed only when the origin state is urgent and has more than one successor, otherwise the filter just forwards all successors without any reduction. In the following we describe a number of optimizations that we included in our implementation.

*Reachable Actions.* In previous sections we have defined  $A = 2^E$ , as the set of actions. This set is unnecessary large and unpractical. The set of *reachable actions* from  $s$  can be semantically defined as  $A_s \stackrel{\text{def}}{=} \{a \in A \mid \exists s', s'' \in S, w \in A^*. s \xrightarrow{w} s' \xrightarrow{a} s''\}$ . Our goal is to compute the smallest set  $A_s^\sharp$  such that  $A_s \subseteq A_s^\sharp \subseteq A$ . Computing a small set has the advantage that potentially less dependencies are introduced, additionally it will reduce the computation time of stubborn sets. We implemented a static analysis in order to compute the set  $A_s^\sharp$ . Our analysis exploits the fact that time can not elapse at a state  $s$ , and thus actions that require a delay to become enabled need not be included in  $A_s^\sharp$ . For the performance sake, the approximation  $A_s^\sharp$  is computed in two steps. The first step is prior to state exploration and is only executed once. In this step for each edge we compute the set of edges it can reach without doing a delay operation. The starting edge is assumed to be enabled and thus we start

with all possible clock assignments in conjunction with the source invariant. If clocks are compared against constants, we add the constraints. Otherwise, if integer variables appear on the guards, we relax (loose) all the information on the affected clock. The second step is executed at every urgent state and it is done by using precomputed data structures from the previous step that collect for every enabled edge the set of edges it can reach and then composes them into actions.

*Broadcast Channels.* Many UPPAAL models use broadcast channels, however the set of possible broadcast synchronizations is exponentially large in terms of the number of potential receivers (in contrast to linear complexity of handshake synchronizations) and hence untenable for larger networks. Instead of computing all possible synchronizations, we compute one super-action for each broadcast sending edge, combining all potential receiving edges from other processes—this serves as a safe over-approximation. Such combined treatment avoids exponential blowup of broadcast actions at the cost of overly-conservative dependency checks, which considers a super-set of associated variables instead of precise sets involving a particular subset of receiving edges. In addition to broadcast synchronizations, the static analysis also supports arrays and C-like structures by expanding them into individual variables. Array indices, references and functions calls are over-approximated by using the ranges from variable types.

*Precomputed Data Structures.* To make our implementation fast, we precompute a number of data structures required by our technique. Examples include, edges leading to locations, some property base sets, reachable edges from locations. In particular, in order to compute the dependence between actions, the associated variable sets are also precomputed in advance for each action. These variable sets are then used to construct a dependency matrix over all reachable actions, thus making the action dependency check a constant-time lookup during verification.

## 6 Experiments

Table 1 shows the results of our POR implementation applied on a number of industrial case studies<sup>1</sup>. The experiments were run on a cluster with AMD EPYC 7551 processor with the timeout of 10 hours (and 15GB of RAM) for all models except for SecureRideSharing where the timeout was 48 hours (and 200GB of RAM). The model instances are suffixed with a number indicating the increasing amount of parallel components (sensors and the like).

FireAlarm is a simplified version of IndustFireAlarm [11] for the communication protocol of a wireless sensor network from German Company SeCa GmbH as described in Section 1. The AGless300 corresponds to a requirement from EN-54 standard that a sensor failure is reported in less than 300 seconds. A stricter property AGless100 is added to evaluate the performance when a property does

<sup>1</sup> Reproducibility package <https://github.com/DEIS-Tools/upor>

Model	Query	without POR		with POR		reduction ratio	
		states	time sec.	states	time sec.	states	time
FireAlarm4	AGnotdeadlock	27	<0.01	22	<0.01	1.23	–
FireAlarm20	AGnotdeadlock	1048635	148.41	270	0.01	3883.83	14841
FireAlarm100	AGnotdeadlock	–	OOM	5350	5.18	–	–
IndustFireAlarm13	AGless100*	931496	97.57	24296	2.81	38.34	34.72
IndustFireAlarm15	AGless100*	3684136	571.75	27672	3.84	133.14	148.89
IndustFireAlarm17	AGless100*	14694312	2884.18	31496	5.09	466.55	566.64
IndustFireAlarm19	AGless100*	58734632	15878.47	35768	7.20	1642.1	2205.34
IndustFireAlarm30	AGless100*	–	OOM	67272	27.92	–	–
IndustFireAlarm100	AGless100*	–	OOM	585272	2753.54	–	–
IndustFireAlarm13	AGless300	3731370	439.50	102570	12.73	36.38	34.52
IndustFireAlarm15	AGless300	14742718	2570.36	116862	17.69	126.15	145.30
IndustFireAlarm17	AGless300	58784210	12833.69	132946	23.15	442.17	554.37
IndustFireAlarm19	AGless300	–	OOM	150822	32.83	–	–
IndustFireAlarm30	AGless300	–	OOM	281172	128.08	–	–
IndustFireAlarm100	AGless300	–	OOM	2380752	12715.08	–	–
IndustFireAlarm13	AGnotdeadlock	3731320	388.63	63618	4.96	58.65	78.35
IndustFireAlarm15	AGnotdeadlock	14742668	2215.16	65654	5.68	224.55	389.99
IndustFireAlarm17	AGnotdeadlock	58784160	11202.80	67818	6.47	866.79	1731.50
IndustFireAlarm19	AGnotdeadlock	–	OOM	70110	8.00	–	–
IndustFireAlarm30	AGnotdeadlock	–	OOM	85004	17.85	–	–
IndustFireAlarm100	AGnotdeadlock	–	OOM	270504	530.46	–	–
SecureRideSharing6	AGlessMaxFail	200141	2.23	200141	5.60	1	0.40
SecureRideSharing7	AGlessMaxFail	7223770	95.60	7223770	252.61	1	0.38
SecureRideSharing8	AGlessMaxFail*	85622469	1467.49	85622469	3691.46	1	0.40
SecureRideSharing9	AGlessMaxFail*	1961298623	43548.8	1961298623	106223.46	1	0.41
SecureRideSharing6	AGnotdeadlock	200141	3.05	184973	6.3	1.08	0.48
SecureRideSharing7	AGnotdeadlock	7223770	122.29	2428033	93.21	2.98	1.31
SecureRideSharing8	AGnotdeadlock	97539581	2058.40	39387328	1845.46	2.48	1.12
SecureRideSharing9	AGnotdeadlock	–	OOM	944892374	55481.09	–	–
TTAC4	AGnotdeadlock	12213203	308.40	11414483	379.51	1.07	0.81
TTAC5	AGnotdeadlock	217259289	6724.25	204152089	8679.56	1.06	0.77
TTPA6	AGnotdeadlock	668421	27.30	668421	55.82	1	0.49
TTPA7	AGnotdeadlock	3329080	166.34	3329080	337.06	1	0.49
TTPA8	AGnotdeadlock	18073077	1096.79	18073077	2229.04	1	0.49
FB14	AGnotdeadlock	98310	138.22	98310	139.5	1	0.99
FB15	AGnotdeadlock	196614	698.54	196614	702.61	1	0.99
FB16	AGnotdeadlock	393222	2794.58	393222	2788.83	1	1

Table 1: Experimental results. Satisfiability results agree for all queries. Queries with \* were not satisfied. The reduction is the ratio of performance without POR and with POR. OOM indicates out of memory.

not hold. Results show exponentially increasing savings in both number of states and computation time.

The SecureRideSharing models a fault-tolerant, duplicate-sensitive aggregation protocol for wireless sensor networks [12,3]. This case study did not show reductions until special treatment for broadcast synchronizations and variable increments was implemented. The AGnotdeadlock property shows substantial reductions, and one instance times out when POR is not used, however for the AGlessMaxFail query the state space is not reducible and the verification time is more than doubled due to variables reverenced in the query.

The TTAC models a Timed Triggered Architecture protocol [13] used in drive-by-wire vehicles. The TTPA models a Time-Triggered Protocol for SAE

class A sensor/actuator networks [27]. The model FB models the Field Buss scheduling protocol [25]. These case studies were selected as they do not allow for any state space reduction, thus allowing us to observe the time-overhead of our method. This overhead varies from almost no overhead for the FB models to twice as slow for the TTPA models.

## 7 Conclusion

We presented an application of partial order reduction based on stubborn sets to the model of network of timed automata in the UPPAAL style, including a detailed analysis of both clock and discrete variable dependencies among the different components. The method allows us to reduce the state space in the situations where a sequence of mutually independent actions is performed while the network is in an urgent configuration where time cannot elapse (caused by the fact that at least one component is in urgent/committed location or there is a clock invariant imposing the urgency). Our method is implemented in the tool UPPAAL and the experiments confirm that for the models with enough independent concurrent behavior in urgent situations, we can achieve exponential speedup in the reachability analysis. For models with limited urgent behavior, the overhead of our method is still acceptable (with the worst-case ratio of about 0.4 slowdown). These results are highly encouraging, yet further optimizations can be achieved by a more detailed static analysis of independent actions, one of the directions for future research.

*Acknowledgments.* We thank Christian Herrera and Sergio Feo Arenis for providing the models we use in our experimental section.

## References

1. L. Aceto, A. Ingólfssdóttir, K. Larsen, and J. Srba. *Reactive Systems: Modelling, Specification and Verification*, page 195. Cambridge University Press, 2007.
2. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, Apr. 1994.
3. S. F. Arenis and B. Westphal. Formal verification of a parameterized data aggregation protocol. In G. Brat, N. Rungta, and A. Venet, editors, *NASA FM*, volume 7871 of *LNCS*, pages 428–434. Springer, 2013.
4. C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.
5. G. Behrmann, P. Bouyer, K. G. Larsen, and R. Pelánek. Lower and upper bounds in zone-based abstractions of timed automata. *STTT*, 8(3):204–215, 2006.
6. G. Behrmann, A. David, and K. G. Larsen. A tutorial on UPPAAL. In M. Bernardo and F. Corradini, editors, *SFM-RT 2004*, number 3185 in *LNCS*, pages 200–236. Springer, 2004.
7. J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi. Partial order reductions for timed systems. In *CONCUR’98*, volume 1466 of *LNCS*, pages 485–500. Springer, 1998.
8. F. Boenneland, P. Jensen, K. Larsen, M. Muniz, and J. Srba. Start pruning when time gets urgent: Partial order reduction for timed systems. In *CAV’18*, volume 10981 of *LNCS*, pages 527–546. Springer-Verlag, 2018.

9. M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis. The IF toolset. In *Int. School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM-RT'04)*, volume 3185 of *LNCS*, pages 237–267. Springer, 2004.
10. D. Dams, R. Gerth, B. Knaack, and R. Kuiper. Partial-order reduction techniques for real-time model checking. *Formal Asp. Comput.*, 10(5-6):469–482, 1998.
11. S. Feo-Arenis, B. Westphal, D. Dietsch, M. Muñoz, and A. S. Andisha. *The Wireless Fire Alarm System: Ensuring Conformance to Industrial Standards through Formal Verification*, pages 658–672. Springer International Publishing, Cham, 2014.
12. S. Gabriel, D. Mosse, J. Brustoloni, and R. Melhem. Ridesharing: Fault tolerant aggregation in sensor networks using corrective actions. the 3rd annual. *IEEE SECON*, 2:595 – 604, 10 2006.
13. K. Godary. *Validation temporelle de réseaux embarqués critiques et faibles pour l'automobile*. PhD thesis, Ins. National des Sc. Appliquées de Lyon, France, 2004.
14. P. Godefroid, J. v. Leeuwen, J. Hartmanis, G. Goos, and P. Wolper. *Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem*, volume 1032 of *LNCS*. Springer-Verlag, 1996.
15. H. Hansen, S. Lin, Y. Liu, T. K. Nguyen, and J. Sun. Diamonds are a girl's best friend: Partial order reduction for timed automata with abstractions. In *CAV'14*, volume 8559 of *LNCS*, pages 391–406. Springer, 2014.
16. C. Herrera, B. Westphal, S. Feo-Arenis, M. Muñoz, and A. Podelski. Reducing quasi-equal clocks in networks of timed automata. In *FORMATS*, number 7595 in *LNCS*, pages 155–170. Springer, Heidelberg, 2012.
17. L. M. Kristensen, K. Schmidt, and A. Valmari. Question-guided stubborn set methods for state properties. *FM in System Design*, 29(3):215–251, 2006.
18. X. Liu and S. A. Smolka. Simple linear-time algorithms for minimal fixed points. In *Automata, Languages and Programming*, pages 53–66. Springer, 1998.
19. D. Lugiez, P. Niebert, and S. Zennou. A partial order semantics approach to the clock explosion problem of timed automata. *Theor. Comput. Sci.*, 345(1):27–59, 2005.
20. M. Minea. Partial order reduction for model checking of timed automata. In *CONCUR'99*, volume 1664 of *LNCS*, pages 431–446. Springer, 1999.
21. M. Muñoz, B. Westphal, and A. Podelski. Detecting quasi-equal clocks in timed automata. In *FORMATS*, pages 198–212, 2013.
22. P. Niebert and H. Qu. Adding invariants to event zone automata. In *FORMATS'06*, volume 4202 of *LNCS*, pages 290–305. Springer, 2006.
23. D. Peled. All from one, one for all: Nn model checking using representatives. In *(CAV'93)*, volume 697 of *LNCS*, pages 409–423. Springer, 1993.
24. M. Perin and J. Faure. Coupling timed plant and controller models with urgent transitions without introducing deadlocks. In *17th International Conference on Emerging Technologies & Factory Automation (ETFA'12)*, pages 1–9. IEEE, 2012.
25. N. Petalidis. Verification of a fieldbus scheduling protocol using timed automata. *Computing and Informatics*, 28:655–672, 01 2009.
26. R. B. Salah, M. Bozga, and O. Maler. On interleaving in timed automata. In *CONCUR'06*, *LNCS*, pages 465–476, 2006.
27. W. Steiner and W. Elmenreich. Automatic recovery of the TTP/A sensor/actuator network. In *WISES*, 2003.
28. A. Valmari and H. Hansen. Stubborn set intuition explained. In *Transactions on Petri Nets and Other Models of Concurrency XII*, pages 140–165. Springer, 2017.



## A Over Approximation for Reachable Actions

### A.1 Abstract Zone Successor

Since we do not have a state we do not know the current integer variable assignments, thus the abstract successor will relax the zone for all clocks. The abstract zone successor  $succ^\sharp : L \times B^+(X) \times E \rightarrow B^+(X)$  takes a location a zone, and an edge and returns a a zone. Formally:  $succ^\sharp(l, Z, (l, h, \phi, \psi, r, l')) = Z'$  where

$$Z' = \begin{cases} (Z \cap I(l) \cap \phi)[r] \cap I(l') & \text{if } V(\phi) = V(I(l)) = V(I(l')) = V(r) = \emptyset \\ \top & \text{otherwise} \end{cases}$$

### A.2 Reachable Edges for an Enabled Edge

Given an enabled edge i.e.  $e = (l, h, \phi, \psi, r, l') \in E$  such that  $e \in a$  for some  $a \in \text{En}(s)$  for some  $s$ . We define a function  $\text{RE} : L \times B^+(X) \times E \rightarrow 2^E$  from a location, a zone, and an edge to a set of edges it could reach without time delays. The set of reachable edges for an enabled edge  $e$  with initial zone  $Z = \top$  is defined as the least fixed point of the following system of equations:

$$\text{RE}(l, Z, e \equiv (l, h, \phi, \psi, r, l')) = \{e \mid \text{if } Z \neq \emptyset\} \cup_{e' = (l', h', \phi', \psi', r', l'') \in E} \text{RE}(l', succ^\sharp(l', Z, e'), e') \quad (1)$$

We have implemented an efficient algorithm for computing the local fixed point for a given location. Our Algorithm is an instantiation of the algorithm proposed by Liu & Smolka [18]. Finally, we construct the set  $A_s^\sharp$  by taking the union of all reachable edges for every component union all possible synchronizations among reachable edges.

**Theorem 4.** *The set  $A_s^\sharp$  is an over approximation of the set of reachable actions, formally:  $A_s \subseteq A_s^\sharp$ .*

*Algorithm for computing reachable edges* Equation 1 stabilizes when a global fixpoint is reached, i.e. a fixpoint for every edge in a component  $\mathcal{A}_i$  is reached. For our purposes we only need the fixpoint for the current edge. Algorithm 2 computes a local fixpoint for a given location. The Algorithm is our instantiation of the algorithm proposed by Liu & Smolka [18]. It maintains a waiting list  $W$  and a list  $P$  of passed states.

Finally we can build the set  $A_s^\sharp$  by composing the set of all reachable edges into actions.

**Definition 12 (Reachable Actions for a Network).** *Given a state  $s = (\vec{l}, Z, \nu) \in S$  and a set of syntactic reachable edges for every component in a*

**Algorithm 2** Computing Reachable Edges for an enabled edge

---

**Input** Enabled edge  $e = (l, h, \phi, \psi, r, l')$ ,  
**Output** Fixpoint of reachable edges for enabled edge  $e$   $\text{RE}(l, Z = \top, e)$

- 1:  $\text{RE} := \emptyset, P := \emptyset, W := \{(l', \text{succ}^\sharp(l, \top, e))\}$
- 2: **while**  $W \neq \emptyset$  **do**
- 3:   Pick  $(l, Z) \in W$  and let  $W := W \setminus \{(l, Z)\}$
- 4:   **if**  $\forall (l, Z') \in P. Z \not\subseteq Z'$  **then**
- 5:      $P := P \cup (l, Z)$
- 6:     **for all**  $e = (l, h, \phi, \psi, r, l') \in E$  **do**
- 7:        $Z' := \text{succ}^\sharp(l, Z, e)$
- 8:       **if**  $Z' \neq \emptyset$  **then**
- 9:           $\text{RE} := \text{RE} \cup \{e\}$
- 10:         $W := W \cup \{(l', Z')\}$
- 11: **return**  $\text{RE}$ ;

---

network  $\text{RE}(s, M_i, \vec{l}^i)$ , the set of syntactic reachable actions for a network is defined as:

$$A_s^\sharp = \bigcup_{i=1}^n \{\{e\} \mid e = (l, \tau, \phi, \psi, r, l') \in \text{RE}(s, M_i, \vec{l}^i)\} \cup \\ \bigcup_{i=1}^n \{\{e_i, e_j\} \mid e_i = (l_i, h_i?, \phi_i, \psi_i, r_i, l'_i) \in \text{RE}(s, M_i, \vec{l}^i) \wedge \\ \exists j \in \{1, \dots, n\}. e_j = (l_j, h_j?, \phi_j, \psi_j, r_j, l'_j) \in \text{RE}(s, M_j, \vec{l}^j)\}$$

## B Proofs

### B.1 Proof of Lemma 1

*Proof (Lemma 1).* Since we are given a variable valuation  $\nu$ , expressions in  $\phi^\nu, (\phi')^\nu$ , and  $r^\nu$  are evaluated to constants in  $\mathbb{N}$ . In this proof for readability we use  $\phi, \phi', r$  instead of  $\phi^\nu, (\phi')^\nu$ , and  $r^\nu$  and we now use  $Z$  for a zone  $\llbracket Z \rrbracket$ .

Statement (1). ( $\subseteq$ ) By structural induction on  $\phi$ . As I.H. let the claim hold for subformulas of  $\phi$ . Let  $\mu[r] \in (Z \cap \phi)[r]$ .

- Case  $\phi \equiv x \bowtie \text{expr}$  with  $\bowtie \in \{<, \leq, \geq, >\}$ .  
Case  $x := \text{expr}_1 \in r$  we have  $x := \text{expr}_1 \rightsquigarrow_s^\sharp \phi$  a contradiction.  
Case  $x := \text{expr}_1 \notin r$  and  $y := \text{expr}_2 \in r$  for some  $y \in X$  with  $x \neq y$  then  $\mu[r](x) = \mu(x)$  and since  $\mu \in Z \cap \phi$  we have  $\mu[r] \in \phi$ .
- Inductive step  $\phi \equiv \phi_1 \wedge \phi_2$  and  $\mu[r] \in (Z \cap \phi_1 \cap \phi_2)[r]$ . By I.H we have  $\mu[r] \in Z[r] \cap \phi_1$  and  $\mu[r] \in Z[r] \cap \phi_2$  therefore  $\mu[x] \in Z[r] \cap \phi_1 \cap \phi_2$ .

( $\supseteq$ ) By structural induction on  $\phi$ . As I.H. let the claim hold for subformulas of  $\phi$ . Let  $\mu[r] \in Z[r] \cap \phi$ .

- Case  $\phi \equiv x \bowtie \text{expr}$  with  $\bowtie \in \{<, \leq, \geq, >\}$   
Case  $x := \text{expr}_1 \in r$  we have  $x := \text{expr}_1 \rightsquigarrow_s^\sharp \phi$  a contradiction.  
Case  $x := \text{expr}_1 \notin r$  and  $y := \text{expr}_2 \in r$  for some  $y \in X$  with  $x \neq y$  then  $\mu[r](x) = \mu(x)$  and since  $\mu \in Z[r] \cap \phi$  we have  $\mu[r] \in \phi$ .

- Inductive step  $\phi \equiv \phi_1 \wedge \phi_2$  and  $\mu[r] \in (Z \cap \phi_1 \cap \phi_2)[r]$ . By I.H. we have  $\mu[r] \in Z[r] \cap \phi_1$  and  $\mu[r] \in Z[r] \cap \phi_2$  therefore  $\mu[x] \in Z[r] \cap \phi_1 \cap \phi_2$ .

Statement (2). Assume  $(Z \cap \phi)[r] \cap \phi \neq \emptyset$ . Show  $(Z \cap \phi)[r] \cap \phi = (Z \cap \phi)[r]$ . The  $\subseteq$  direction follows trivially since  $(Z \cap \phi)[r] \cap \phi \subseteq (Z \cap \phi)[r]$ . We now proof the  $\supseteq$  direction by structural induction on  $\phi$ . As I.H. let the claim hold for subformulas of  $\phi$ . Let  $\mu[r] \in (Z \cap \phi)[r]$ .

- Case  $\phi \equiv x \bowtie \text{expr}$  and  $x := \text{expr}_1 \notin r$ . Then  $\mu[r](x) = \mu(x)$  and since  $\mu \in Z \cap \phi$  we have  $\mu[r] \in \phi$ .
- Case  $\phi \equiv x \bowtie \text{expr}$  and  $x := \text{expr}_1 \in r$ . Consider the following cases:
  - $\text{expr}^\nu = \text{expr}'_1$ . Then  $\mu[r](x) = \mu(x)$  and  $\mu[r] \in \phi$ .
  - $\text{expr}^\nu < \text{expr}'_1$ . Consider the cases:
    - \*  $x \bowtie \text{expr}$  with  $\bowtie \in \{<, \leq\}$  then  $(Z \cap \phi)[r] \cap \phi \neq \emptyset$ .
    - \*  $x \bowtie \text{expr}$  with  $\bowtie \in \{>, \geq\}$  then  $\mu[r] \in \phi$
  - $\text{expr}^\nu > \text{expr}'_1$ . Consider the cases:
    - \*  $x \bowtie \text{expr}$  with  $\bowtie \in \{<, \leq\}$  then  $\mu[r] \in \phi$ .
    - \*  $x \bowtie \text{expr}$  with  $\bowtie \in \{>, \geq\}$  then  $(Z \cap \phi)[r] \cap \phi \neq \emptyset$ .
- Inductive step  $\phi \equiv \phi_1 \wedge \phi_2$  and  $\mu[r] \in (Z \cap \phi_1 \cap \phi_2)[r]$ . By I.H. we have  $\mu[r] \in (Z \cap \phi_1)[r] \cap \phi_1$  and  $\mu[r] \in (Z \cap \phi_2)[r] \cap \phi_2$ . Therefore  $\mu[r] \in \phi_1 \cap \phi_2$  and  $\mu[r] \in (Z \cap \phi_1 \cap \phi_2)[r] \cap \phi_1 \cap \phi_2$ .

Statement (3). By structural induction on  $\phi$  and  $\phi'$ . As I.H. let the claim hold for subformulas of  $\phi$  and  $\phi'$ .

- Case  $\phi \equiv x \bowtie \text{expr}$  and  $\phi' \equiv x \bowtie \text{expr}_1$ . Since  $\phi \rightsquigarrow_s^\# \phi'$  we have  $|\Gamma_x(Z)| = 1$ . By assumption we are given  $\mu \in Z$ ,  $\mu \in \phi'$ , and  $Z \cap \phi \neq \emptyset$ . If  $\mu \in \phi$  we are done. Otherwise  $\mu \notin \phi$  which implies that  $\mu(x) \not\models \phi$ , but since  $Z \cap \phi \neq \emptyset$  there must exist a  $\mu' \in Z$  s.t.  $\mu'(x) \models \phi$ , that is  $\mu(x) \neq \mu'(x)$ . Yielding  $|\Gamma_x(Z)| \neq 1$ , a contradiction.
- Case  $\phi \equiv x \bowtie \text{expr}$  and  $\phi' \equiv y \bowtie \text{expr}_1$ . Since  $\phi \rightsquigarrow_s^\# \phi'$  we have  $|\Gamma_x(Z)| = 1$  and  $|\Gamma_y(Z)| = 1$ . In addition we have  $Z \cap \phi \neq \emptyset$ . Let  $\mu' \in Z \cap \phi$ , then  $\mu' \in \phi$  and since  $|\Gamma_x(Z)| = 1$  we have that all valuations in  $Z$  are in  $\phi$  i.e.  $Z \subseteq \phi$  which is equivalent to  $Z = Z \cap \phi$ . By assumption  $\mu \in Z$  thus  $\mu \in Z \cap \phi$ .
- Inductive step: given  $\phi$  and  $\phi' \equiv \phi'_1 \wedge \phi'_2$ . By assumption  $\mu \in \phi'_1 \wedge \phi'_2$ . Then  $\mu \in \phi'_1$  and by I.H.  $\mu \in Z \cap \phi'_1$ . Also  $\mu \in \phi'_2$  and by I.H.  $\mu \in Z \cap \phi'_2$ . Therefore,  $\mu \in Z \cap \phi'_1 \cap \phi'_2$ .
- Inductive step: given  $\phi \equiv \phi_1 \wedge \phi_2$  and  $\phi'$ . By assumption  $\mu \in \phi'$  and  $Z \cap \phi_1 \cap \phi_2 \neq \emptyset$ . By I.H. we have  $\mu \in Z \cap \phi_1$  and  $\mu \in Z \cap \phi_2$ . Therefore  $\mu \in Z \cap \phi_1 \cap \phi_2$ .

Statement (4). ( $\subseteq$ ) By structural induction on  $\phi$ . As I.H. let the claim hold for subformulas of  $\phi$  and  $\phi'$ .

- Case  $\phi \equiv x \bowtie \text{expr}$  with  $\bowtie \in \{<, \leq\}$ . Then  $\mu(x) \in Z \cap \phi'$  and  $\mu(x) \leq \text{expr}^\nu$ . Now by contradiction assume  $\mu(x) > \text{expr}^{\nu_1}$  i.e.  $\mu(x) \notin \phi^{\nu_1}$  then this implies  $Z \cap \phi^{\nu_1} = \emptyset$  (a contradiction) because  $|\Gamma_x(Z)| = 1$ .
- Case  $\phi \equiv x \bowtie \text{expr}$  with  $\bowtie \in \{>, \geq\}$ . Symmetric to the previous case.

- Case  $\phi \equiv x \bowtie \text{expr}_1 \wedge y \bowtie \text{expr}_2$  with  $\bowtie \in \{<, \leq, \geq, >\}$ . We have  $\mu \in Z \cap x \bowtie \text{expr}_1^\nu \cap y \bowtie \text{expr}_2^\nu$ . Then  $\mu(x) \in Z \cap x \bowtie \text{expr}_1^\nu$  and  $\mu(y) \in Z \cap y \bowtie \text{expr}_2^\nu$ . By I.H.  $\mu(x) \in Z \cap x \bowtie \text{expr}_1^{\nu_1}$  and  $\mu(y) \in Z \cap y \bowtie \text{expr}_2^{\nu_1}$ . Then  $\mu \in Z \cap x \bowtie \text{expr}_1^{\nu_1} \cap y \bowtie \text{expr}_2^{\nu_1}$ .

The case ( $\subseteq$ ) is symmetric.  $\square$

## B.2 Proof of Theorem 2

*Proof (Theorem 2).* In the following, let  $s = (\vec{l}, Z, \nu)$  and the successors states of  $s$  after  $a_1$  and  $a_2$  be  $s \xrightarrow{a_1} s_1 = (\vec{l}_1, Z_1, \nu_1)$  and  $s \xrightarrow{a_2} s_2 = (\vec{l}_2, Z_2, \nu_2)$ . We use  $\exists i$  as a shorthand for  $\exists i \in \{1, \dots, n\}$ . For the rest of the proof we can assume  $a_1 \rightsquigarrow_s^\# a_2$ . First consider the case when  $a_1 \notin \text{En}(s)$  or  $a_2 \notin \text{En}(s)$ . Then the conditions from Definition 7 trivially hold and we have  $a_1 \rightsquigarrow_s a_2$ .

Consider the more involved case when  $a_1 \in \text{En}(s)$  and  $a_2 \in \text{En}(s)$ .

We show Definition 7 Condition 1, Consider  $s \xrightarrow{a_1} s_1$  we show that  $a_2 \in \text{En}(s_1)$ . Condition (Ind1) implies  $\text{Pre}(a_1) \cap \text{Pre}(a_2) = \emptyset$  i.e.  $\forall e \in a_2. \exists i. \vec{l}_1^i = \text{src}(e)$ , therefore all sources for edges in  $a_2$  are still in  $\vec{l}_1$ . By (Ind2) committed locations do not disable  $a_2$ . By (Ind3) and (Ind4) all guards and invariants of  $a_2$  are enabled after  $a_1$ . (We now devote to demonstrate this).

Without loss of generality let  $a_1 = \{e_i\}$   $a_2 = \{e_j\}$  with  $i \neq j$ ,  $e_i = (l_i, \tau, \phi_i, \psi_i, r_i, l'_i) \in E_i$  and  $e_j = (l_j, \tau, \phi_j, \psi_j, r_j, l'_j) \in E_j$ . Let  $\mu[r_i^\nu] \in Z_1$  i.e.

$$\mu[r_i^\nu] \in (Z \cap I(\vec{l})^\nu \cap (\phi_i)^\nu)[r_i^\nu] \cap (I(\vec{l})[l'_i/l_i])^{\nu_1}.$$

By the semantics of resets we have

$$\mu \in Z \cap I(\vec{l})^\nu \cap \phi_i^\nu.$$

By Lemma 1(3) since  $\phi_i \rightsquigarrow_s^\# \phi_j$ ,  $Z \cap I(\vec{l})^\nu \cap \phi_j^\nu \neq \emptyset$  and  $\mu \in Z \cap I(\vec{l})^\nu \cap \phi_i^\nu$ . We obtain

$$\mu \in Z \cap I(\vec{l})^\nu \cap \phi_i^\nu \cap \phi_j^\nu.$$

By the semantics of reset we have

$$\mu[r_i^\nu] \in (Z \cap I(\vec{l})^\nu \cap \phi_i^\nu \cap \phi_j^\nu)[r_i^\nu].$$

By assumption  $r_i \rightsquigarrow_s^\# \phi_j$  and by using Lemma 1(1) we have

$$(Z \cap I(\vec{l})^\nu \cap \phi_j^\nu \cap \phi_i^\nu)[r_i^\nu] = (Z \cap I(\vec{l})^\nu \cap \phi_i^\nu)[r_i^\nu] \cap \phi_j^\nu,$$

therefore

$$\mu[r_i^\nu] \in (Z \cap I(\vec{l})^\nu \cap \phi_i^\nu)[r_i^\nu] \cap \phi_j^\nu.$$

and by assumption

$$\mu[r_i^\nu] \in (Z \cap I(\vec{l})^\nu \cap \phi_i^\nu)[r_i^\nu] \cap I(\vec{l}[l'_i/l_i])^{\nu_1} \cap \phi_j^\nu.$$

Since  $r_i \rightsquigarrow_s^\# \phi_j$  we have that  $V(r_i) \cap V(\phi_j) = \emptyset$  which means that  $\nu|_{V(\phi_j)} = \nu_1|_{V(\phi_j)}$  i.e. valuations agree for all discrete variables in  $\phi_j$ , which implies  $\phi_j^\nu = \phi_j^{\nu_1}$ . Otherwise if  $V(r_i) \cap V(\phi_j) \neq \emptyset$  since  $r_i \rightsquigarrow_s^\# \phi_j$  we have the case that exists  $v \in V(r_i)$  and  $r_i$  contains the expression  $v := v + 1$  an increment (or decrement) and  $\phi_j$  contains a constraint of the form  $x \leq \text{expr}$  (or  $x \geq \text{expr}$  for decrements) where  $v$  occurs in  $\text{expr}$ , then note  $\nu(v) \leq \nu_1(v)$  (or  $\nu(v) \geq \nu_1(v)$  in the case of decrements) then we have  $\phi_j^\nu \subseteq \phi_j^{\nu_1}$  and thus

$$\mu[r_i^\nu] \in (Z \cap I(\vec{l})^\nu \cap \phi_i^\nu)[r_i^\nu] \cap I(\vec{l}'_i/l_i)^{\nu_1} \cap \phi_j^{\nu_1}. \quad (2)$$

Which means that the guard of  $a_2$  is satisfied after executing  $a_1$ . Let  $\nu_{12} = \nu_1[r_j^{\nu_1}]$  be the variable valuation after executing  $a_2$  from  $s_1$ . Now, we need to show that the destination invariant of  $a_2$  is also satisfied. We need to show  $\mu[r_i^\nu][r_j^{\nu_1}] \in ((Z \cap I(\vec{l})^\nu \cap \phi_i^\nu)[r_i^\nu] \cap I(\vec{l}'_i/l_i) \cap \phi_j^{\nu_1})[r_j^{\nu_1}] \cap I(\vec{l}'_i/l_i, l'_j/l_j)^{\nu_{12}}$ . By the semantics of reset and continuing from 2 we have

$$\mu[r_i^\nu][r_j^{\nu_1}] \in ((Z \cap I(\vec{l})^\nu \cap \phi_i^\nu)[r_i^\nu] \cap I(\vec{l}'_i/l_i)^{\nu_1} \cap \phi_j^{\nu_1})[r_j^{\nu_1}]$$

Note that  $I(\vec{l}'_i/l_i)^{\nu_1} = \bigcap_k^{\{1, \dots, n\} \setminus \{i\}} I(\vec{l}^k)^{\nu_1} \cap I(l'_i)^{\nu_1}$ . By assumption  $r_j \rightsquigarrow_s^\# I(\vec{l}^k)$  for  $k \in \{1, \dots, n\} \setminus \{j\}$  and  $r_j \rightsquigarrow_s^\# I(l'_i)$ . By Lemma 1(1) we obtain:

$$\begin{aligned} & ((Z \cap I(\vec{l})^\nu \cap \phi_i^\nu)[r_i^\nu] \cap I(\vec{l}'_i/l_i)^{\nu_1} \cap \phi_j^{\nu_1})[r_j^{\nu_1}] \\ &= ((Z \cap I(\vec{l})^\nu \cap \phi_i^\nu)[r_i^\nu] \cap I(l_j)^{\nu_1} \cap \phi_j^{\nu_1})[r_j^{\nu_1}] \bigcap_k^{\{1, \dots, n\} \setminus \{i, j\}} I(\vec{l}^k)^{\nu_1} \cap I(l'_i)^{\nu_1} \end{aligned}$$

By assumption  $r_j \rightsquigarrow_s^\# I(\vec{l}^k)$  for  $k \in \{1, \dots, n\} \setminus \{j\}$  and  $r_j \rightsquigarrow_s^\# I(l'_i)$ . Which implies (as above) that  $\nu_1$  and  $\nu_{12}$  agree on all discrete variables in all invariants except possible for some variables which have been incremented or decremented (c.f. above for 2 ) thus we have

$$\begin{aligned} & ((Z \cap I(\vec{l})^\nu \cap \phi_i^\nu)[r_i^\nu] \cap I(l_j)^{\nu_1} \cap \phi_j^{\nu_1})[r_j^{\nu_1}] \bigcap_k^{\{1, \dots, n\} \setminus \{i, j\}} I(\vec{l}^k)^{\nu_1} \cap I(l'_i)^{\nu_1} \\ & \subseteq ((Z \cap I(\vec{l})^\nu \cap \phi_i^\nu)[r_i^\nu] \cap I(l_j)^{\nu_1} \cap \phi_j^{\nu_1})[r_j^{\nu_1}] \bigcap_k^{\{1, \dots, n\} \setminus \{i, j\}} I(\vec{l}^k)^{\nu_{12}} \cap I(l'_i)^{\nu_{12}} \end{aligned}$$

Therefore

$$\mu[r_i^\nu][r_j^{\nu_1}] \in ((Z \cap I(\vec{l})^\nu \cap \phi_i^\nu)[r_i^\nu] \cap I(l_j)^{\nu_1} \cap \phi_j^{\nu_1})[r_j^{\nu_1}] \bigcap_k^{\{1, \dots, n\} \setminus \{i, j\}} I(\vec{l}^k)^{\nu_{12}} \cap I(l'_i)^{\nu_{12}}. \quad (3)$$

Then we need to show that  $\mu[r_i^\nu][r_j^{\nu_1}] \in I(l'_j)^{\nu_{12}}$ . By assumption  $(Z \cap I(\vec{l})^\nu \cap (\phi_j)^\nu)[r_j^\nu] \cap (I(\vec{l})[l'_j/l_j])^{\nu_2} \neq \emptyset$ . From Derivation 3 we obtain

$$\mu[r_i^\nu][r_j^{\nu_1}] \in ((Z \cap I(\vec{l})^\nu \cap \phi_i^\nu)[r_i^\nu] \cap \phi_j^{\nu_1})[r_j^{\nu_1}]$$

Since  $r_i \rightsquigarrow_s^\# \phi_j$  by Lemma 1(1) we obtain

$$\mu[r_i^\nu][r_j^{\nu_1}] \in ((Z \cap I(\vec{l})^\nu \cap \phi_i^\nu \cap \phi_j^{\nu_1})[r_i^\nu])[r_j^{\nu_1}]$$

We have that  $r_i \rightsquigarrow_s^\# r_j$ . In particular the only allowed shared variables in the updates are the integer variables being incremented (decremented) in both

updates and for every zone  $Z'$  and any valuations  $\nu_i, \nu_j$  since the updates do not share clocks. we have  $Z'[r_i^{\nu_i}][r_j^{\nu_j}] = Z'[r_j^{\nu_j}][r_i^{\nu_i}]$ . Therefore we obtain

$$\mu[r_j^{\nu_1}][r_i^{\nu_1}] \in ((Z \cap I(\vec{l})^\nu \cap \phi_i^\nu \cap \phi_j^{\nu_1})[r_j^{\nu_1}])[r_i^{\nu_1}]$$

By the semantics of resets we have

$$\mu[r_j^{\nu_1}] \in (Z \cap I(\vec{l})^\nu \cap \phi_i^\nu \cap \phi_j^{\nu_1})[r_j^{\nu_1}]$$

Since  $r_i \not\sim_s^\# r_j$  clock are not shared in updates and  $\nu_1 = \nu[r_i^{\nu_1}]$  we have  $r_j^{\nu_1}$  and  $r_j^{\nu_1}$  agree on clock updates (for any zone  $Z$  we have  $Z[r_j^{\nu_1}] = Z[r_j^{\nu_1}]$ ). Further  $\phi_i \not\sim_s^\# r_j$  we have  $\phi_i^{\nu_1} \subseteq \phi_i^{\nu_1}$  (because of incs, decs) (then for any zone  $Z \cap \phi_i^{\nu_1} \supseteq Z \cap \phi_i^{\nu_1}$ ). Thus

$$\mu[r_j^{\nu_1}] \in (Z \cap I(\vec{l})^\nu \cap \phi_i^\nu \cap \phi_j^{\nu_1})[r_j^{\nu_1}] \quad (4)$$

By using Lemma 1(3) and using zone  $Z'$  as template from the Equation 4 without constraint  $\phi_i$  and since by assumption  $Z' \cap I(\vec{l}[l'_j/l_j])^{\nu_2} \neq \emptyset$ . We have that  $\mu[r_j^{\nu_1}] \in I(\vec{l}[l'_j/l_j])^{\nu_2}$  and in particular  $\mu[r_j^{\nu_1}] \in I(l'_j)^{\nu_2}$ . Since  $r_i \not\sim_s^\# I(l'_j)$  no clock in  $I(l'_j)$  is updated by  $r_i$  and we have  $\mu[r_j^{\nu_1}][r_i^{\nu_1}] \in I(l'_j)^{\nu_2}$  i.e.  $\mu[r_j^{\nu_1}][r_i^{\nu_1}] \in I(l'_j)^{\nu_2}$ . In particular we have that  $I(l'_j)^{\nu_2} \subseteq I(l'_j)^{\nu_{12}}$  (because of possible incs and decs in  $r_i$ ). Thus we have that  $\mu[r_i^{\nu_1}][r_j^{\nu_1}] \in I(l'_j)^{\nu_{12}}$ . Finally we obtain

$$\mu[r_i^{\nu_1}][r_j^{\nu_1}] \in ((Z \cap I(\vec{l})^\nu \cap \phi_i^\nu)[r_i^{\nu_1}] \cap I(l_j)^{\nu_1} \cap \phi_j^{\nu_1})[r_j^{\nu_1}] \cap I(l'_i)^{\nu_{12}} \cap I(l'_j)^{\nu_{12}}.$$

That is  $a_2 \in \text{En}(s_1)$  if the variable guards are satisfied. The case for variables is easier, since the only shared variables are the ones in increments or decrements. The case for  $s \xrightarrow{a_2} s_2$  and  $a_1 \in \text{En}(s_2)$  is symmetric. Therefore we conclude that Definition 7 Statement 1 hold.

We now show Definition 7 Condition 2,  $s \xrightarrow{a_1 a_2} s'$  iff  $s \xrightarrow{a_2 a_1} s'$ . Since Definition 7 Statement 1 holds, we have  $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} s'_1$  and  $s \xrightarrow{a_2} s_2 \xrightarrow{a_1} s'_2$ . We need to show that  $s'_1 = s'_2$ . Let  $s'_1 = (\vec{l}'_1, Z'_1, \nu'_1)$  and  $s'_2 = (\vec{l}'_2, Z'_2, \nu'_2)$ . First consider  $a_1 = \{(l_i, \tau, \phi_i, \psi_i, r_i, l'_i)\}$  and  $a_2 = \{(l_j, \tau, \phi_j, \psi_j, r_j, l'_j)\}$ . In particular note that since  $\text{Pre}(a_1) \cap \text{Pre}(a_2) = \emptyset$  we have that  $l_i \neq l_j$ .

We show that  $\vec{l}'_1 = \vec{l}'_2$ , by staring at  $\vec{l}$  and executing  $a_1$  we obtain  $\vec{l}[l'_i/l_i]$  and by executing  $a_2$  we obtain  $\vec{l}[l'_i/l_i, l'_j/l_j]$ . By starting at  $\vec{l}$  and executing  $a_2$  we obtain  $\vec{l}[l'_j/l_j]$  and by executing  $a_1$  we obtain  $\vec{l}[l'_j/l_j, l'_i/l_i]$ . Since  $i \neq j$  the substitutions occur at different positions in  $\vec{l}$  and  $\vec{l}[l'_j/l_j, l'_i/l_i] = \vec{l}[l'_i/l_i, l'_j/l_j]$ .

We show that  $\nu'_1 = \nu'_2$ . Assume  $\nu'_1 \neq \nu'_2$  Then a variable is modified by  $a_1$  and  $a_2$ , but the only shared variables are when both actions are either increments or decrements of a variable. Since increments and decrements commute we have  $\nu'_1 = \nu'_2$ .

Now we show that  $Z'_1 = Z'_2$ . Note that  $Z = Z \cap I(\vec{l})$ . We have the following:

$$Z'_1 = ((Z \cap \phi_i^\nu)[r_i^{\nu_1}] \cap I(\vec{l}[l'_i/l_i])^{\nu_1} \cap \phi_j^{\nu_1})[r_j^{\nu_1}] \cap I(\vec{l}[l'_i/l_i, l'_j/l_j])^{\nu_1} \neq \emptyset$$

By definition of invarinat we have

$$Z'_1 = ((Z \bigcap_{k=1}^n I(\bar{l}^k)^\nu \cap \phi_i^\nu)[r_i^\nu] \bigcap_{k \in \{1, \dots, n\} \setminus \{i\}} I(\bar{l}^k)^{\nu_1} \cap I(l'_i)^{\nu_1} \cap \phi_j^{\nu_1}[r_j^{\nu_1}] \cap I(\bar{l}[l'_i/l_i, l'_j/l_j])^{\nu_1})^{\nu_1}$$

Note that because  $\bigcap_{k \in \{1, \dots, n\} \setminus \{i\}} I(\bar{l}^k) \rightsquigarrow_s^\# r_i$  and  $\nu_1 = \nu[r_i^\nu]$  we have that  $\phi \equiv \bigcap_{k \in \{1, \dots, n\} \setminus \{i\}} I(\bar{l}^k)$  we have  $\phi^\nu \subseteq \phi^{\nu_1}$ . By using Lemma 1 (4) we have

$$Z'_1 = ((Z \bigcap_{k=1}^n I(\bar{l}^k)^\nu \cap \phi_i^\nu)[r_i^\nu] \bigcap_{k \in \{1, \dots, n\} \setminus \{i\}} I(\bar{l}^k)^\nu \cap I(l'_i)^{\nu_1} \cap \phi_j^{\nu_1}[r_j^{\nu_1}] \cap I(\bar{l}[l'_i/l_i, l'_j/l_j])^{\nu_1})^{\nu_1}$$

by Lemma 1(2) we obtain

$$Z'_1 = ((Z \bigcap_{k=1}^n I(\bar{l}^k)^\nu \cap \phi_i^\nu)[r_i^\nu] \cap I(l'_i)^{\nu_1} \cap \phi_j^{\nu_1}[r_j^{\nu_1}] \cap I(\bar{l}[l'_i/l_i, l'_j/l_j])^{\nu_1})^{\nu_1}$$

since  $\phi_j \rightsquigarrow_s^\# r_i$  by Lemma 1 (4) we have

$$Z'_1 = ((Z \bigcap_{k=1}^n I(\bar{l}^k)^\nu \cap \phi_i^\nu)[r_i^\nu] \cap I(l'_i)^{\nu_1} \cap \phi_j^{\nu_1}[r_j^{\nu_1}] \cap I(\bar{l}[l'_i/l_i, l'_j/l_j])^{\nu_1})^{\nu_1}$$

by Lemma 1(1)

$$Z'_1 = ((Z \bigcap_{k=1}^n I(\bar{l}^k)^\nu \cap \phi_i^\nu \cap \phi_j^\nu)[r_i^\nu] \cap I(l'_i)^{\nu_1} \cap \phi_j^{\nu_1}[r_j^{\nu_1}] \cap I(\bar{l}[l'_i/l_i, l'_j/l_j])^{\nu_1})^{\nu_1}$$

by Lemma 1(1)

$$Z'_1 = ((Z \bigcap_{k=1}^n I(\bar{l}^k)^\nu \cap \phi_i^\nu \cap \phi_j^\nu)[r_i^\nu][r_j^{\nu_1}] \cap I(l'_i)^{\nu_1} \cap I(\bar{l}[l'_i/l_i, l'_j/l_j])^{\nu_1})^{\nu_1}$$

since  $r_i \rightsquigarrow_s^\# r_j$

$$Z'_1 = ((Z \bigcap_{k=1}^n I(\bar{l}^k)^\nu \cap \phi_i^\nu \cap \phi_j^\nu)[r_j^{\nu_1}][r_i^\nu] \cap I(l'_i)^{\nu_1} \cap I(\bar{l}[l'_i/l_i, l'_j/l_j])^{\nu_1})^{\nu_1}$$

Note that  $I(\bar{l}[l'_i/l_i, l'_j/l_j])^{\nu_1}$  contains  $I(l'_i)^{\nu_1}$  and since  $\nu'_1 = \nu_1[r_j]$  with  $I(l'_i) \rightsquigarrow_s^\# r_j$  we have  $I(l'_i)^{\nu_1} \subseteq I(l'_i)^{\nu'_1}$ . By using Lemma 1 (4) we have

$$Z'_1 = ((Z \bigcap_{k=1}^n I(\bar{l}^k)^\nu \cap \phi_i^\nu \cap \phi_j^\nu)[r_j^{\nu_1}][r_i^\nu] \cap I(\bar{l}[l'_i/l_i, l'_j/l_j])^{\nu_1})^{\nu_1}$$

Note that  $I(\bar{l}[l'_i/l_i, l'_j/l_j])^{\nu_1}$  contains  $I(l'_j)^{\nu_1}$

$$Z'_1 = ((Z \bigcap_{k=1}^n I(\bar{l}^k)^\nu \cap \phi_i^\nu \cap \phi_j^\nu)[r_j^{\nu_1}][r_i^\nu] \cap I(l'_j)^{\nu_1} \cap I(\bar{l}[l'_i/l_i, l'_j/l_j])^{\nu_1})^{\nu_1}$$

In the following let  $r|_X$  be the sequence of clock updates obtained from  $r$ . Since  $r_j \not\rightsquigarrow_s^\# r_i$  it is the case that  $r_j^\nu|_X = r_j^{\nu_1}|_X$ , Otherwise if  $r_j^\nu|_X \neq r_j^{\nu_1}|_X$  we have that there exists an update  $x := \text{expr}$  with  $\text{expr}^\nu \neq \text{expr}^{\nu_1}$  and some variable  $v \in \text{Read}(x := \text{expr})$  being modified by  $r_i$ . By definition we have that  $v \in \text{Write}(r_i) \cup \text{Inc}(r_i) \cup \text{Dec}(r_i)$  and  $r_j \rightsquigarrow_s^\# r_i$  a contradiction thus  $r_j^\nu|_X = r_j^{\nu_1}|_X$  and we obtain

$$Z'_1 = ((Z \bigcap_{k=1}^n I(\vec{l}^k)^\nu \cap \phi_i^\nu \cap \phi_j^\nu)[r_j^\nu][r_i^\nu] \cap I(l'_j)^{\nu_1} \cap I(\vec{l}[l'_i/l_i, l'_j/l_j])^{\nu_1})^{\nu_1}$$

by Lemma 1(1)

$$Z'_1 = ((Z \bigcap_{k=1}^n I(\vec{l}^k)^\nu \cap \phi_j^\nu)[r_j^\nu] \cap \phi_i^\nu[r_i^\nu] \cap I(l'_j)^{\nu_1} \cap I(\vec{l}[l'_i/l_i, l'_j/l_j])^{\nu_1})^{\nu_1}$$

Since  $r_j \rightsquigarrow_s^\# r_i$  it is the case that  $r_i^\nu|_X = r_i^{\nu_2}|_X$ , Otherwise if  $r_i^\nu|_X \neq r_i^{\nu_2}|_X$  we obtain a contradiction as above thus

$$Z'_1 = ((Z \bigcap_{k=1}^n I(\vec{l}^k)^\nu \cap \phi_j^\nu)[r_j^\nu] \cap \phi_i^\nu[r_i^{\nu_2}] \cap I(l'_j)^{\nu_1} \cap I(\vec{l}[l'_i/l_i, l'_j/l_j])^{\nu_1})^{\nu_1}$$

Note that  $\nu'_1 = \nu'_2$  thus

$$Z'_1 = ((Z \bigcap_{k=1}^n I(\vec{l}^k)^\nu \cap \phi_j^\nu)[r_j^\nu] \cap \phi_i^\nu[r_i^{\nu_2}] \cap I(l'_j)^{\nu_2} \cap I(\vec{l}[l'_i/l_i, l'_j/l_j])^{\nu_2})^{\nu_2}$$

We have  $I(l'_j) \rightsquigarrow_s^\# r_i$  and  $\nu'_2 = \nu_2[r_i^{\nu_2}]$  then  $I(l'_j)^{\nu_2} \subseteq I(l'_j)^{\nu_2}$  thus by Lemma 1 (4) we have

$$Z'_1 = ((Z \bigcap_{k=1}^n I(\vec{l}^k)^\nu \cap \phi_j^\nu)[r_j^\nu] \cap \phi_i^\nu[r_i^{\nu_2}] \cap I(l'_j)^{\nu_2} \cap I(\vec{l}[l'_i/l_i, l'_j/l_j])^{\nu_2})^{\nu_2}$$

by Lemma 1(1)

$$Z'_1 = ((Z \bigcap_{k=1}^n I(\vec{l}^k)^\nu \cap \phi_j^\nu)[r_j^\nu] \cap I(l'_j)^{\nu_2} \cap \phi_i^\nu[r_i^{\nu_2}] \cap I(\vec{l}[l'_i/l_i, l'_j/l_j])^{\nu_2})^{\nu_2}$$

We have  $\phi_i \rightsquigarrow_s^\# r_j$  and  $\nu_2 = \nu[r_j^\nu]$  then  $\phi_i^\nu \subseteq \phi_i^{\nu_2}$  thus by Lemma 1 (4) we have

$$Z'_1 = ((Z \bigcap_{k=1}^n I(\vec{l}^k)^\nu \cap \phi_j^\nu)[r_j^\nu] \cap I(l'_j)^{\nu_2} \cap \phi_i^{\nu_2}[r_i^{\nu_2}] \cap I(\vec{l}[l'_i/l_i, l'_j/l_j])^{\nu_2})^{\nu_2}$$

Since  $\bigcap_k^{\{1, \dots, n\} \setminus \{j\}} I(\vec{l}^k) \rightsquigarrow_s^\# r_j$  by Lemma 1(1) we have

$$Z'_1 = ((Z \cap I(\vec{l}^j)^\nu \cap \phi_j^\nu)[r_j^\nu] \bigcap_k^{\{1, \dots, n\} \setminus \{j\}} I(\vec{l}^k)^\nu \cap I(l'_j)^{\nu_2} \cap \phi_i^{\nu_2}[r_i^{\nu_2}] \cap I(\vec{l}[l'_i/l_i, l'_j/l_j])^{\nu_2})^{\nu_2}$$



we have  $\nu_2 = \nu[r_j^\nu]$  and  $\phi \equiv \bigcap_k^{\{1, \dots, n\} \setminus \{j\}} I(\vec{l}^k)$  we have  $\phi^\nu \subseteq \phi^{\nu_2}$ . By using Lemma 1 (4) we have

$$Z'_1 = ((Z \cap I(\vec{l}^j)^\nu \cap \phi_j^\nu)[r_j^\nu] \bigcap_k^{\{1, \dots, n\} \setminus \{j\}} I(\vec{l}^k)^{\nu_2} \cap I(l_j^{\nu_2}) \cap \phi_i^{\nu_2})[r_i^{\nu_2}] \cap I(\vec{l}[l'_i/l_i, l'_j/l_j])^{\nu_2}$$

Finally, by grouping the invariants we get

$$Z'_1 = ((Z \cap \phi_j^\nu)[r_j^\nu] \cap I(\vec{l}[l'_j/l_j])^{\nu_2} \cap \phi_i^{\nu_2})[r_i^{\nu_2}] \cap I(\vec{l}[l'_i/l_i, l'_j/l_j])^{\nu_2} = Z'_2$$

Therefore we conclude that Definition 7 Statement 2 holds.  $\square$

### B.3 Proof of Corollary 1

*Proof (Corollary 1).* By induction on the length  $n$  of  $w$ . As I.H. let the claim hold for lengths  $< n$ . Base  $|w| = 0$ . By definition of  $A'$  we have that  $a \rightsquigarrow_{s'}^\# a'$ . Since  $s' = s$  we have  $a \rightsquigarrow_{s'}^\# a'$ . Inductive step  $|w| = n$ . Let  $w = a_1 \dots a_n$ . We need to show  $s_0 \xrightarrow{a_1} \dots \xrightarrow{a_{n-1}} s_{n-1} \xrightarrow{a_n} s' \xrightarrow{a'} s'' \implies a \rightsquigarrow_{s'}^\# a'$ . As I.H. we have  $a \rightsquigarrow_{s_{n-1}}^\# a'$ . In addition we have that  $a \rightsquigarrow_{s_0}^\# a_n$ . We need to show that conditions (Ind1), (Ind2), (Ind3), and (Ind4) hold at state  $s'$  for  $a$  and  $a'$ . Conditions (Ind1), (Ind2) are state independent and hold for state  $s'$ . We need to show that conditions (Ind3) and (Ind4) hold for  $s'$ . Conditions (Ind3) and (Ind4) use Definition 8 Independence of Operations. Conditions in Definition 8 are state independent except for Condition (7) which is  $\{x \mid x \in X(\text{op}_1) \cup X(\text{op}_2) \text{ and } |\Gamma_x(Z)| \neq 1 \text{ and } \text{op}_1, \text{op}_2 \in B(X, V)\} = \emptyset$ . This means that we only need to show that Condition (7) holds at state  $s_n$ . Without loss of generality let  $a = \{(l_1, \tau, \phi_1, \psi_1, r_1, l'_1)\}$ ,  $a' = \{(l_2, \tau, \phi_2, \psi_2, r_2, l'_2)\}$ ,  $a_n = \{(l_n, \tau, \phi_n, \psi_n, r_n, l'_n)\}$ ,  $s_{n-1} = (\vec{l}_{n-1}, Z_{n-1}, \nu_{n-1})$ , and,  $s_n = (\vec{l}_n, Z_n, \nu_n)$ .

By I.H. we have  $|\Gamma_x(Z_{n-1})| = 1$  for any clock in  $x \in X(\text{Guard}(\phi_1)) \cup X(\text{Guard}(\phi_2)) \cup X(\bigcap\{I(\vec{l}_0^j) \mid \vec{l}^j \notin \text{Pre}(i) \text{ and } 1 \leq j \leq n\})$  for  $i \in \{a, a'\}$ .

Let  $x$  by a clock in  $x \in X(\text{Guard}(\phi_1)) \cup X(\text{Guard}(\phi_2)) \cup X(\bigcap\{I(\vec{l}_0^j) \mid \vec{l}^j \notin \text{Pre}(i) \text{ and } 1 \leq j \leq n\})$  for  $i \in \{a, a'\}$ . Note that reset  $r_n$  can not cause  $|\Gamma_x(Z_{n-1})| \neq 1$ . In addition, intersections with guards  $\phi_n$  and  $I(l_n)$  can not increase the valuations of  $x$  causing  $|\Gamma_x(Z_n)| > 1$ , because it will imply that  $Z_{n-1} \cap \phi_n \supset Z_{n-1}$ . Therefore, we have  $|\Gamma_x(Z_n)| = 1$ .  $\square$

### B.4 Proof of Lemma 2

*Proof (Lemma 2).* By induction on  $w$ . As I.H. assume that the claim holds for all sequences of length smaller and equal to the one of  $w$ .

Base case  $|w| = 0$  then  $s = s'$  and  $\text{zt}(s')$ .

Inductive step  $|w'a'| = n + 1$  Assume  $s_0 \xrightarrow{w'} s_n \xrightarrow{a'} s' \wedge \text{zt}(s_0)$ . By I.H. we have that  $\text{zt}(s_i)$  for  $0 \leq i \leq n$ . By definition we have  $\text{Delay}_{s_0}^\# = \{a\} \cup \{a' \in A_s^\# \mid a \rightsquigarrow_{s_0}^\# a'\}$ . By assumption we have  $\text{tea}^\#(a, s_0)$  with  $a \in \text{En}(s_0)$  and  $a \rightsquigarrow_{s_0}^\# a'$ .

From Corollary 1 we obtain  $a \rightsquigarrow_{s_i}^\# a'$  for  $i \leq n$ . Therefore  $a \in \text{En}(s')$ . Let  $s_0 = (\vec{l}, Z, \nu)$  and  $s' = (\vec{l}', Z', \nu')$  by definition of  $\text{tea}^\#(a, s_0)$  we have to consider two cases:

- Case  $\exists l \in \text{Pre}(a)$ .  $l \in L^u \cup L^c$ . By Definition 9 and condition (Ind1) we have  $\exists i. \vec{l}^i \in \text{Pre}(a) \wedge (\vec{l}')^i \in L^u \cup L^c$  (i.e. the urgent/committed location stopping time is still in  $\vec{l}'$ ). By the semantics of the transition relation we have  $\text{zt}(s')$ .
- Case  $\exists l \in \text{Pre}(a)$ .  $\forall \mu \in Z, d \in \mathbb{R}_{\geq 0}$ .  $\mu + d \models I(l) \implies d = 0$ . Since Condition (Ind1) holds, we have that  $\exists i. (\vec{l}')^i \in \text{Pre}(a)$  such that  $I((\vec{l}')^i)$  is stopping time. We continue by contradiction assume  $\forall l \in \text{Pre}(a)$ .  $\exists \mu' \in Z', d \in \mathbb{R}_{\geq 0}$ .  $\mu' + d \models I(l) \wedge d > 0$ . This implies that the invariant  $I((\vec{l}')^i)$  has been relaxed by a clock reset in some action  $a''$  in  $w'a'$ . By definition we have  $\text{Delay}_{s_0}^\# \subseteq \{a' \in A_s^\# \mid \exists l \in \text{Pre}(a), r \in \text{Update}(a'). X(r) \in X(I(l))\}$  implied by the dependency relation  $\rightsquigarrow_{s_0}^\#$  and such an  $a''$  can not be in  $w'a'$ .  $\square$

## B.5 Lemma 4

**Lemma 4.** *Given basic formula  $\varphi$ , the following holds:  $\forall s, s' \in S, a \in (A_s^\# \setminus \varphi^\#)$ .  $s \xrightarrow{a} s' \wedge s \not\models \varphi \implies s' \not\models \varphi$*

*Proof.* By structural induction on  $\varphi$ . Let  $s = (\vec{l}, Z, \nu)$  and  $s' = (\vec{l}', Z', \nu')$ . For the I.H. let the proposition hold for all subformulas of  $\varphi$ . We continue by the base cases:

- $\varphi \equiv l$ , then for all  $i \in \{1, \dots, n\}$  we have  $\vec{l}^i \neq l$ . Assume  $(\vec{l}', Z', \nu') \models l$ , then there exist  $i \in \{1, \dots, n\}$  such that  $(\vec{l}')^i = l$ , by definition of transition relation we have either  $\vec{l}' = \vec{l}[l'_i/l_i]$  or  $\vec{l}' = \vec{l}[l'_j/l_j, l'_i/l_i]$  if there exists edges  $e_i = (l_i, \tau, \phi, \psi, r, l'_i) \in E_i$  or  $e_i = (l_i, c?, \phi_i, \psi_i, r_i, l'_i) \in E_i$  and  $e_j = (l_j, c!, \phi_j, \psi_j, r_j, l'_j) \in E_j$ . Since  $l = l'_i$  or  $l = l'_j$ ,  $l \in \text{Post}(a)$  and thus  $a \in \varphi_s^\#$ .
- $\varphi \equiv \text{deadlock}$ , by definition we have an  $a' \in \text{En}(s)$  in  $\text{deadlock}_s^\#$ . We also have that  $a \rightsquigarrow_s^\# a'$ . By Theorem 2 we have that  $a \rightsquigarrow_{s'}^\# a'$ , and by definition of  $\rightsquigarrow_s$  we have that  $a \in \text{En}(s')$ . Therefore  $s' \not\models \text{deadlock}$ .
- $\varphi \equiv x \bowtie c \bowtie \in \{<, \leq, \geq, >\}$ , we have  $s \not\models x \bowtie c$  and  $s' \models x \bowtie c$ . Since  $s$  time can not elapse with action transitions, this is only possible if there was a reset for  $x$  in  $a$ . Thus we have  $x := \text{expr} \in \text{Update}(a)$  with  $x \bowtie \text{expr}' \rightsquigarrow_s^\# x := \text{expr}$  and thus  $a \in \varphi_s^\#$ .
- $\varphi \equiv \varphi_v$  for  $v \in V$ . By assumption we have  $\nu(v) \not\models \varphi_v$  and  $s' \models \varphi_v$  which means that  $\nu'(v) \models \varphi_v$ . Thus the valuation of  $v$  has been updated which means  $\exists \text{op} \in \text{Op}(a)$ .  $\varphi_v \rightsquigarrow_s^\# \text{op}$ . Therefore,  $a \in \varphi_s^\#$ .
- We continue with the inductive case:  $\varphi \equiv \varphi_1 \wedge \varphi_2$ . Then either  $s \not\models \varphi_1$  or  $s \not\models \varphi_2$ . Without loss of generality let  $s \not\models \varphi_1$ , then  $\varphi_s^\# = (\varphi_1)_s^\#$ . Since  $s' \models \varphi_1 \wedge \varphi_2$ , we have that  $s' \models \varphi_1$  by firing  $a$ . By I.H. we have that  $a \in (\varphi_1)_s^\#$  and thus  $a \in \varphi_s^\#$ .  $\square$

### B.6 Proof of Lemma 3

We proof this lemma with help from Lemma 4.

*Proof (Lemma 3).* We continue by induction on  $w$ . As I.H. assume that the claim holds for all sequences of length smaller and equal to the one of  $w$ .

Base case  $|w| = 0$  then  $s = s'$  and  $s' \not\models \varphi$ .

Inductive Step  $|wa| = n+1$ , we have  $s_0 \xrightarrow{w} s_n \xrightarrow{a} s_{n+1}$ . If exists  $s_i$  in the path from  $s_0$  to  $s_n$  such that  $s_i \models \varphi$  by I.H. we are done. Therefore let  $s_n \not\models \varphi$  and  $s_{n+1} \models \varphi$ . By Lemma 4 we have that  $a \in \varphi_{s_n}^\sharp$ . We need to show that  $a \in \varphi_{s_0}^\sharp$ . For this we need to consider the structure of the formula to infer their corresponding sets. We continue by structural induction on  $\varphi$ . As I.H. let  $a \in \varphi_{s_0}^\sharp$  for all sub formulas of  $\varphi$ .

- Basis  $\varphi \equiv l$ ,  $\varphi \equiv x = y$ ,  $\varphi \equiv x \bowtie c$   $\varphi \equiv \varphi_v$ , the set of actions definition is independent of  $s_0$  (same set for all states) and we have  $\varphi_{s_0}^\sharp = \varphi_{s_n}^\sharp$  and since  $a \in \varphi_{s_n}^\sharp$  we have  $a \in \varphi_{s_0}^\sharp$ .
- Basis  $\varphi \equiv \text{deadlock}$ , If  $a \in \varphi_{s_0}^\sharp$  then we are done. Otherwise we have  $a \notin \varphi_{s_0}^\sharp$ . By definition of  $\varphi_{s_0}^\sharp$  we have  $\exists a' \in \varphi_{s_0}^\sharp$  such that  $a' \in \text{En}(s_0)$  and  $a' \rightsquigarrow_{s_0}^\sharp a''$  for all  $a''$  in  $wa$ . By Corollary 1 we have  $a' \rightsquigarrow_{s_n}^\sharp a$  and therefore  $a' \in \text{En}(s_{n+1})$  and  $s_{n+1} \not\models \text{deadlock}$  a contradiction. Therefore  $a \in \varphi_{s_0}^\sharp$ .
- Inductive step  $\varphi \equiv \varphi_1 \wedge \varphi_2$ . Then either  $s_n \not\models \varphi_1$  or  $s_n \not\models \varphi_2$ . Without loss of generality let  $s_n \not\models \varphi_1$ , then  $\varphi_{s_n}^\sharp = (\varphi_1)_{s_n}^\sharp$ . By I.H. we have that  $a \in (\varphi_1)_{s_0}^\sharp$ .  $\square$

### B.7 Proof of Theorem 3

*Proof (Theorem 3).*

*Termination* The set  $A_s^\sharp$  is finite and so is the set  $\varphi_s^\sharp \subseteq A_s^\sharp$ . The Algorithm starts with  $\text{St}_s^\sharp := \varphi_s^\sharp$ , in each iteration the size of  $\text{St}_s^\sharp$  can only increase because the only operation applied to  $\text{St}_s^\sharp$  is union. In the worst case we have  $\text{St}_s^\sharp = A_s^\sharp$ .

*Correctness* We show that the algorithm satisfies the conditions of Definition 2.

- *Definition 2 Condition Z* this condition follows from Line 1 where the output is  $\text{En}(s)$ .
- *Definition 2 Condition D* By line 3 there is an action  $a \in \text{St}_s^\sharp$  with  $\text{tea}^\sharp(a, s)$  and  $a \in \text{En}(s)$ . Because of the loop from Line 16 we have  $\{a\} \cup \{a' \in A_s^\sharp \mid a \rightsquigarrow_s^\sharp a'\} \subseteq \text{St}_s^\sharp$ . By using Lemma 2 this condition is satisfied.
- *Definition 2 Condition R* When the algorithm exists the main while loop from Line 5 we have to cases:
  - Case  $\text{En}(s) \cap \text{St}_s^\sharp \neq \text{En}(s)$ , in this case there was no reduction and the condition trivially holds.
  - Case  $W = \emptyset$ , From Line 4 we have that all actions from  $\varphi_s^\sharp$  are in  $W$ . From Line 6 all elements from  $\varphi_s^\sharp$  have been added to  $\text{St}_s^\sharp$ . By applying Lemma 3 the condition is satisfied.

– *Definition 2 Condition  $\mathcal{W}$*   $\forall a \in \text{St}(s), w \in \overline{\text{St}(s)}^* . \text{zt}(s) \wedge s \xrightarrow{wa} s' \implies s \xrightarrow{aw} s'$  Recall that  $\text{St}(s)$  is defined as  $\text{St}_s^\sharp$ . If the while loop from Line 5 terminates with  $\text{En}(s) \cap \text{St}_s^\sharp = \text{En}(s)$  then transitions with  $w$  are not possible and the claim trivially holds. Contrary we have two cases:

- Case  $a \notin \text{En}(s)$ , by contradiction, consider the sequence  $s_0 \xrightarrow{w} s_n \xrightarrow{a'} s_{n+1}$  there exists a transition  $a_i$  in  $wa'$  such that  $a \notin \text{En}(s_i)$  and  $a \in \text{En}(s_{i+1})$ . Let  $s_i = (\vec{l}_i, Z_i, \nu_i)$  and  $s_{i+1} = (\vec{l}_{i+1}, Z_{i+1}, \nu_{i+1})$ . Action  $a$  can become enabled in  $s_{i+1}$  in the following cases:  
Source locations of  $a$  are not in  $\vec{l}_i$  but are in  $\vec{l}_{i+1}$ , this case is captured by Line 11.  
A guard  $g \in \text{Guard}(a)$  becomes enabled, this case is captured by Line 14.  
Therefore, such an action  $a_i$  can not be in  $w$ .
- Case  $a \in \text{En}(s)$ . The following is an invariant of the main while loop in Line 5  $A_s^\sharp = \text{St}^\sharp \cup R$ . In addition the following is an invariant from the for loop in Line 16.

$$\begin{aligned} \forall a \in \text{St}_s^\sharp, a' \in R. a \in \text{En}(s) \wedge \\ ((\text{Pre}(a) \cap \text{Pre}(a') \neq \emptyset) \vee (\text{Parallel}(a, a') \wedge a \rightsquigarrow_s^\sharp a')) \implies a' \in \text{St}_s^\sharp \end{aligned} \quad (5)$$

Let  $w = a_1 a_2 \dots a_n$  and  $s \xrightarrow{a_1} s_1 \dots \xrightarrow{a_n} s_n$ . By Invariant 5 we have that  $a \rightsquigarrow_s^\sharp a_i$  with  $a_i$  in  $w$ . From Corollary 1 we have  $a \rightsquigarrow_s^\sharp a_i$ . This means that  $a$  is independent with all actions in  $w$  and since  $a \in \text{En}(s)$  we can move  $a$  to produce  $s \xrightarrow{aw} s'$ .

□

## B.8 Proof of Theorem 4

*Proof.* By Definition we have that  $A_s \stackrel{\text{def}}{=} \{a \in A \mid \exists s', s'' \in S, w \in A^* . s \xrightarrow{w} s' \xrightarrow{a} s''\}$ . We continue by induction on the length of  $w$ . As I.H. assume that the claim holds for all paths of length at most  $|w|$ . In the Following let  $s = (\vec{l}, Z, \nu)$ .

- Case  $|w| = 0$  then we have  $a = \emptyset$  and  $a \in A_s^\sharp$ .
- Case  $|w| = n + 1$  then we have  $s \xrightarrow{w} s' \xrightarrow{a} s''$ . By I.H. we have that  $a' \in A_s^\sharp$  for all  $a'$  in  $w$ . We continue by case distinction on  $a$ . For the rest of the proof let  $s' = (\vec{l}', Z', \nu')$ .
  - Case  $a \equiv \{e\}$  for some  $e = (l, h, \phi, \psi, r, l') \in E_i$ . Since the transition is executed we have  $Z' \models \phi$  and  $\nu' \models \psi$  and  $Z' \models I(l') \vee \exists x \in X(I(l')). x \in \mathbf{M}$ . Assume that for some  $v \in V$  in  $\psi$  we have  $\nu(v) \neq \nu'(v)$  then we have that  $v \in \mathbf{M}$  or  $v$  is not local and  $a \in A_s^\sharp$ . Otherwise we have  $\nu \models \psi$ . For zone  $Z$ , since time can not elapse by actions in  $w$  we have two cases: (1)  $Z \supseteq Z'$ , because intersections from the guards on  $w$  will only produce subsets of  $Z$ . Since  $Z' \models \phi$  we have  $Z \models \phi$  and  $a \in A_s^\sharp$ . (2)  $Z \not\supseteq Z'$ , assume  $Z \not\models \phi$  but  $Z' \models \phi$  in this case there was a reset to some clock  $x$ . By definition of RE we have  $x \in \mathbf{M}$  and  $a \in A_s^\sharp$ .

- Case  $a \equiv \{e_i, e_j\}$  for some  $e_i \in E_i$  and  $e_j \in E_j$ . By definition of  $\xrightarrow{\{e_i, e_j\}}$  we have  $(\vec{l}, Z, \nu) \xrightarrow{\{e_i, e_j\}} (\vec{l}[l'_j/l_j, l'_i/l_i], Z', \nu')$  if there exists  $e_i = (l_i, h_i!, \phi_i, \psi_i, r_i, l'_i) \in E_i$  and  $e_j = (l_j, h_j?, \phi_j, \psi_j, r_j, l'_j) \in E_j$  s.t.  $h_i^\nu = h_j^\nu$ , and  $Z' = (Z \wedge I(\vec{l})^\nu \wedge \phi_i^\nu \wedge \phi_j^\nu)[r^\nu] \wedge I(\vec{l}[l'_j/l_j, l'_i/l_i])^{\nu'}$ , where  $\llbracket Z' \rrbracket \neq \emptyset$ ,  $\nu \models (\psi_i^\nu \wedge \psi_j^\nu)$ ,  $\nu' = \nu[r_i^\nu \cup r_j^\nu]$ , and if  $\vec{l}^k \in L_k^c$  for some  $1 \leq k \leq n$  then  $l_i \in L_i^c$  or  $l_j \in L_j^c$ . From the definition we have that  $Z' \models \phi_i, \nu \models \psi_i$  and  $Z' \models I(l'_i) \vee \exists x \in X(I(l'_i)).x \in M_i$ . Therefore  $\{e_i\} \in \text{RE}(s, M_i, l_i)$ . Analogously we have  $\{e_j\} \in \text{RE}(s, M_j, l_j)$ . By Definition 12 we have  $a \in A_s^\#$ . □